

Universidad Carlos III de Madrid

Escuela Politécnica Superior



Ingeniería Informática

Proyecto Fin de Carrera

Madrid, Octubre 2012

*Estudio de arquitecturas en soft-procesors y
comparativa de rendimiento y consumo con
procesadores comerciales*

Autor: Víctor Nieto Talaván

Tutores: David Expósito Singh

Oscar Pérez Alonso



Agradecimientos

Este proyecto se lo quiero agradecer de la mejor manera posible a toda la gente que me ha apoyado a lo largo de mi vida, en especial a toda mi familia.

Mi hermano Marcos, mis padres Julio y Mari Carmen, y mi novia Bea forman parte de mi día a día, y sin ellos no habría conseguido seguir adelante. Siempre han estado ahí para todo, para lo bueno y lo malo, y eso es lo mejor que me ha podido pasar.

En segundo lugar se lo quiero agradecer a mis amigos, David, Castre, Fombe, Honorio, Marta y Rocío, por ser capaces de aguantarme y animarme todo este tiempo. Tampoco puedo olvidar los buenos momentos que he pasado con mis compañeros de clase, Champi, Cristian, Sohrab, Alberto, Juanto y un largo etc. de personas que han conseguido hacerme sacar una carcajada y una sonrisa.

Finalmente, me gustaría agradecer enormemente toda la ayuda que he recibido por parte de mis tutores, David y Oscar, por su gran apoyo en este proyecto, sus conocimientos y su paciencia. No olvidaré los buenos momentos pasados con la gente del laboratorio, por todas las risas que he podido compartir con ellos y por conseguir hacer que finalice este proyecto.

*A todos ellos, **gracias de todo corazón.***

“El logro más impresionante de la industria del software es su continua anulación de los constantes y asombrosos logros de la industria del hardware”

Henry Petroski



Resumen

En los últimos 20 años, la tecnología ha evolucionado muy rápidamente, siendo ésta parte de nuestro día a día. Actualmente, tanto los ordenadores como los dispositivos móviles disponen de una capacidad de procesamiento elevada, gracias al uso de **procesadores empotrados**.

Estos tipos de dispositivos se pueden encontrar en una gran variedad de productos que abarcan desde los dispositivos anteriormente mencionados, hasta reproductores multimedia, televisores o cámaras, incluyendo grandes sistemas implantados en aviones y automóviles. Para todas estas aplicaciones los sistemas empotrados necesitan ofrecer una gran eficiencia en la relación entre la **potencia de cómputo y el consumo eléctrico**.

Para este tipo de sistemas, es común el uso de FPGAs como sistemas de estudio de arquitecturas. El motivo es su reducido coste y la enorme flexibilidad que ofrece el hardware reconfigurable. Típicamente una **FPGA** destina parte de sus recursos a la implementación de bloques o módulos hardware para el procesamiento de señales. Sin embargo, la parte de control se suele ejecutar mediante software en la propia FPGA. El modo de conseguirlo es implementar un procesador simple en la FPGA, denominado **soft-processor** y programarlo.

En este proyecto, se tratará de conseguir una plataforma de trabajo con el mejor rendimiento estimado, y la mejor relación rendimiento/consumo amén de otras características como la utilización de la FPGA. Esto se realizará mediante el análisis y desarrollo de distintos **benchmarks** (pruebas de rendimiento) para diversas arquitecturas.

Para finalizar se validará la solución encontrada a través del estudio anterior con **procesadores empotrados comerciales** de potencia similar.



Abstract

In the past 20 years, technology has evolved rapidly, and it became part of our daily life. Currently, both computers and mobile devices have a high processing capability by means of **embedded processors**.

These devices can be found in a wide variety of products, ranging from the aforementioned devices, to media players, TVs and cameras, including large systems implemented in aircraft and automobiles. For all these applications, embedded systems need to provide high efficiency and good ratios between computing **performance and power consumption**.

For such systems, it is common to use FPGAs as a test platform for evaluating different system architectures. The reason is its low cost and enormous flexibility offered by reconfigurable hardware. **FPGAs** typically devote part of their resources to the implementation of hardware blocks or modules for signal processing. However, the control part is usually performed by FPGA's own software. The way to achieve this is to implement a single processor in the FPGA, called **soft-processor** and program it.

In this project, we will try to develop and analyze a FPGA-based test platform with the best estimated performance and the best performance/consumption ratio. Additionally, we evaluate other features such as the use facility. This will be done through the development, evaluation and analysis of different **benchmarks** for various architectures.

Finally, the obtained results will be compared against other **commercial embedded processors** of similar power.



Índice

1	Introducción y objetivos	21
1.1	Motivación del proyecto	21
1.2	Objetivos del proyecto	22
1.3	Contenido de la memoria	23
2	Estado del Arte	25
2.1	FPGAs	25
2.1.1	Xilinx	26
2.1.2	Altera	28
2.2	Soft-Processors	29
2.2.1	Arquitectura MicroBlaze	30
2.2.2	Arquitectura Nios II	31
2.2.3	Arquitectura LEON 4	33
2.2.4	Arquitectura OpenRISC 1000	35
2.3	Arquitecturas principales	36
2.3.1	Arquitectura x86	37
2.3.2	Arquitectura MIPS	38
2.3.3	Arquitectura ARM	40
2.4	Pruebas de rendimiento	41
3	Análisis	45
3.1	Elección de plataforma de trabajo	45
3.2	Descripción General	49
3.2.1	Capacidades generales	49
3.2.2	Restricciones generales	50



3.2.3	Entorno operacional	51
3.3	Requisitos de usuario.....	52
3.3.1	Requisitos de Capacidad	53
3.3.2	Requisitos de Restricción	55
3.4	Medidas de rendimiento	57
3.4.1	Capacidad de cómputo	58
3.4.2	Acceso a memoria	58
3.4.3	Consumo de energía	59
4	Diseño	61
4.1	Arquitectura de los benchmarks	61
4.2	Diseño detallado	62
4.2.1	Dhrystone 2.1	63
4.2.2	Linpack.....	65
4.2.3	Whetstone.....	68
4.2.4	VitiJumps	70
4.2.5	VitiRAM.....	73
4.3	Método de trabajo	76
5	Implementación	77
5.1	Digilent Nexys3 y Atlys	77
5.2	Nintendo DS	85
5.3	Sony Playstation Portable.....	87
5.4	Linux, Microsoft DOS y Microsoft Windows NT	88
6	Evaluación y análisis de resultados	91
6.1	Primera iteración.....	91
6.2	Segunda iteración	104



6.3	Tercera iteración	116
6.4	Cuarta iteración.....	121
7	Planificación y presupuesto	135
7.1	Planificación	135
7.2	Presupuesto.....	138
8	Conclusiones y trabajos futuros	140
8.1	Conclusiones.....	141
8.2	Líneas de investigación futuras	143
9	Acrónimos y abreviaturas	147
10	Bibliografía	151
	Anexo A: Material entregado	155
	Anexo B: Resumen de arquitecturas implementadas	157



Índice de Tablas

Tabla 1: Modelos principales de las FPGA de Xilinx	26
Tabla 2: Entorno operacional Digilent	51
Tabla 3: Entorno operacional arquitectura x86	51
Tabla 4: Entorno operacional arquitectura MIPS	51
Tabla 5: Entorno operacional arquitectura ARM	52
Tabla 6: Plantilla de requisitos	52
Tabla 7: RUC-01	53
Tabla 8: RUC-02	53
Tabla 9: RUC-03	53
Tabla 10: RUC-04	54
Tabla 11: RUC-05	54
Tabla 12: RUC-06	54
Tabla 13: RUC-07	54
Tabla 14: RUC-08	55
Tabla 15: RUC-09	55
Tabla 16: RUR-01	55
Tabla 17: RUR-02	55
Tabla 18: RUR-03	56
Tabla 19: RUR-04	56
Tabla 20: RUR-05	56
Tabla 21: RUR-06	56
Tabla 22: RUR-07	57
Tabla 23: Operaciones y tipos en Dhrystone 2.1	64

Tabla 24: Operaciones y tipos en DGEFA de Linpack 100x100	66
Tabla 25: Operaciones y tipos en DGESE de Linpack 100x100.....	66
Tabla 26: Condiciones ejecutadas en el primer módulo de VitiJumps.....	71
Tabla 27: Placas de desarrollo Digilent Nexys3 y Digilent Atlys.....	77
Tabla 28: Resultados Nexys3 v001	92
Tabla 29: Ocupación de la FPGA en Nexys3 v001 y v002.....	93
Tabla 30: Unidades de multiplicación y división de enteros en Nexys3 v005, v006 y v007	94
Tabla 31: Resultados para Nexys3 v005, v006 y v007	94
Tabla 32: Resultados para Nexys3 v009 y v010	96
Tabla 33: Frecuencias de procesador de Nexys3 v019, v024, v025, v026 y v027.	102
Tabla 34: Tamaños de caché para Nexys3 v019 y v024, y Atlys v001, v002 y v003	104
Tabla 35: Políticas de caché en Atlys v002 y v007.....	109
Tabla 36: Rendimiento en Atlys v002 y v007.....	109
Tabla 37: Configuración del predictor de saltos en Atlys v002, v008, v009 y v010	112
Tabla 38: Comparativa de rendimiento energético en Dhrystone 2.1	133
Tabla 39: Comparativa de rendimiento energético en Whetstone.....	133
Tabla 40: Comparativa de rendimiento energético en Linpack.....	134
Tabla 41: Costes de personal	138
Tabla 42: Costes de hardware.....	139
Tabla 43: Costes de software.....	140
Tabla 44: Presupuesto final	140
Tabla 45: Resumen de configuraciones hardware en Digilent Nexys3.....	158
Tabla 46: Resumen de configuraciones hardware en Digilent Atlys	158



Tabla 47: Resumen de configuraciones de caché en Digilent Nexys3.....	159
Tabla 48: Resumen de configuraciones de caché en Digilent Atlys	160



Índice de Figuras

Ilustración 1: Arquitectura del procesador MicroBlaze	31
Ilustración 2: Arquitectura del procesador Nios II	33
Ilustración 3: Arquitectura del procesador LEON 4.....	34
Ilustración 4: Arquitectura del procesador OpenRISC 1200.....	35
Ilustración 5: Arquitectura del procesador MIPS32 M14K.....	39
Ilustración 6: Arquitectura de un procesador ARM11	41
Ilustración 7: Diagrama de componentes de la placa Digilent Nexys3	46
Ilustración 8: Diagrama de componentes de la placa Digilent Atlys	47
Ilustración 9: Ejemplo de sistema empotrado desarrollado en el Xilinx Platform Studio.....	48
Ilustración 10: Ejemplo de producción de software en el Xilinx Software Development Kit	49
Ilustración 11: Arquitectura estándar para los benchmarks.....	61
Ilustración 12: Ejemplo de ejecución de Dhrystone 2.1	65
Ilustración 13: Ejemplo de ejecución de Linpack 100x100	67
Ilustración 14: Ejemplo de ejecución de Whetstone	70
Ilustración 15: Ejemplo de ejecución de VitiJumps	72
Ilustración 16: Funcionamiento de la evaluación de rendimiento de escritura en memoria.....	73
Ilustración 17: Funcionamiento de la evaluación de rendimiento de copia en memoria.....	74
Ilustración 18: Ejemplo de ejecución de VitiRAM	75
Ilustración 19: Base System Builder de Xilinx Platform Studio	78
Ilustración 20: Parámetros básicos configurables del soft-processor MicroBlaze ...	79

Ilustración 21: Configuración de dispositivos necesarios para la arquitectura MicroBlaze	80
Ilustración 22: Configuración de memoria caché para la arquitectura MicroBlaze .	81
Ilustración 23: Opciones avanzadas del procesador MicroBlaze	82
Ilustración 24: Entorno de desarrollo Xilinx Software Development Kit para el procesador MicroBlaze.....	83
Ilustración 25: Ventana de selección de sistema base para la plataforma MicroBlaze	84
Ilustración 26: Configuración de los timers en el XilKernel.....	84
Ilustración 27: Ejemplo de proyecto en Programmer's Notepad para Nintendo DS	86
Ilustración 28: Ejemplo de compilación de proyecto para Sony Playstation Portable	87
Ilustración 29: Ejemplo de compilación de proyecto en OpenWatcom	89
Ilustración 30: Ocupación en la FPGA en Nexys3 v005, v006 y v007.....	95
Ilustración 31: Ocupación de la FPGA en Nexys3 v008, v009 y v010.....	97
Ilustración 32: Benchmark Dhrystone 2.1 para comparativa de tamaños de caché en Nexys3.....	98
Ilustración 33: Benchmark Linpack para comparativa de tamaños de caché en Nexys3.....	99
Ilustración 34: Benchmark Whetstone para comparativa de tamaños de caché en Nexys3.....	99
Ilustración 35: Ocupación de FPGA en Nexys3 con distintos tamaños de caché ..	100
Ilustración 36: Ocupación de bloques BRAM en Nexys3 para distintos tamaños de caché	101
Ilustración 37: Escalabilidad de Nexys3 en términos de frecuencia (Dhrystone)..	102
Ilustración 38: Escalabilidad de Nexys3 en términos de frecuencia (Linpack)	103
Ilustración 39: Resultado de Dhrystone 2.1 para Nexys3 v019 y v024, y Atlys v001, v002 y v003	105
Ilustración 40: Resultado de Linpack para Nexys3 v019 y v024, y Atlys v001, v002 y v003	105

Ilustración 41: Resultado de Whetstone para Nexys3 v019 y v024, y Atlys v001, v002 y v003	106
Ilustración 42: Resultado de VitiRAM en escritura para Nexys3 v024 y Atlys v001	107
Ilustración 43: Resultado de VitiRAM en copia para Nexys3 v024 y Atlys v001.	107
Ilustración 44: Resultado de VitiJumps para Nexys3 v024 y Atlys v001	108
Ilustración 45: Rendimiento relativo entre Atlys v002 y Atlys v007	110
Ilustración 46: Resultado de VitiRAM en escritura para Atlys v002 y Atlys v007	110
Ilustración 47: Resultado de VitiRAM en copia para Atlys v002 y Atlys v007	111
Ilustración 48: Resultado de VitiJumps para Atlys v002, v008, v009 y v010	112
Ilustración 49: Resultado de Dhrystone 2.1 para Nexys3 v024 y Atlys v002, v007 y v017	113
Ilustración 50: Resultado de Linpack para Nexys3 v024 y Atlys v002, v007 y v017	114
Ilustración 51: Resultado de Whetstone para Nexys3 v024 y Atlys v002, v007 y v017	114
Ilustración 52: Resultado de VitiRAM en escritura para Nexys3 v024, y Atlys v002, v007 y Atlys v017	115
Ilustración 53: Resultado de VitiRAM en copia para Nexys3 v024, y Atlys v002, v007 y Atlys v017	115
Ilustración 54: Resultado de Dhrystone 2.1 con distintas optimizaciones generales de GCC	117
Ilustración 55: Resultado de Linpack con distintas optimizaciones generales de GCC	118
Ilustración 56: Resultado de Whetstone con distintas optimizaciones generales de GCC	118
Ilustración 57: Resultado de Dhrystone 2.1 con distintas optimizaciones específicas de GCC	119
Ilustración 58: Resultado de Linpack con distintas optimizaciones específicas de GCC	119



Ilustración 59: Resultado de Whetstone con distintas optimizaciones específicas de GCC.....	120
Ilustración 60: Resultados de Dhrystone 2.1 para distintas arquitecturas	122
Ilustración 61: Resultados de Linpack en precisión simple para distintas arquitecturas.....	123
Ilustración 62: Resultados de Linpack de precisión doble para distintas arquitecturas	124
Ilustración 63: Resultados de Whetstone para distintas arquitecturas.....	125
Ilustración 64: Resultados de VitiRAM en escritura para distintas arquitecturas ..	126
Ilustración 65: Resultados de VitiRAM en copias de datos para distintas arquitecturas.....	126
Ilustración 66: Resultados de VitiJumps en saltos directos para distintas arquitecturas.....	128
Ilustración 67: Resultados de VitiJumps en saltos indirectos para distintas arquitecturas.....	130
Ilustración 68: Vatímetro Watts Up? .Net.....	131
Ilustración 69: Planificación del proyecto	137
Ilustración 70: Implementación dudosa del XilKernel.....	142
Ilustración 71: Error grave de ejecución en el XilKernel	142

1 Introducción y objetivos

La finalidad de este capítulo es dar a conocer el contenido de este documento, y permitir un primer encuentro con este proyecto de final de carrera. Además se detallará la motivación de la realización del proyecto y los principales objetivos que se plantean.

Para finalizar este punto se detallará el contenido de cada capítulo de la memoria.

1.1 Motivación del proyecto

Este proyecto nació por la necesidad de conseguir una plataforma para sistemas empuados, que fuese lo suficientemente potente y genérica para cualquier tipo de sistema a desarrollar.

En la actualidad, los sistemas empuados están basados en diseños completamente orientados para una tarea específica, por lo que su coste de desarrollo, tanto a nivel de hardware como de software, puede llegar a ser demasiado elevado si la complejidad del mismo también es elevada.

Las soluciones más económicas en la actualidad son aquellas basadas en microcontroladores. Un ejemplo de estos, son los sistemas basados en Arduino, que incluyen un pequeño microcontrolador y entradas y salidas analógicas y digitales. El microcontrolador que incluyen los Arduinos es un Atmel AVR de 8 bits, a una frecuencia de 16 MHz y con una memoria RAM de trabajo de 2 Kb en adelante. Con estas características limitadas es difícil desarrollar aplicaciones computacionalmente costosas, por lo que los sistemas empuados que incluyan microcontroladores deben de ejecutar un software simple y de bajo coste de cómputo.

Por otro lado, esta falta de potencia computacional dificulta en gran medida el procesamiento de las señales de entrada/salida, por lo que, o bien se añade hardware externo que solucione dicha problemática, o bien se utiliza un sistema más complejo. Las FPGAs son la alternativa perfecta, puesto que permiten realizar un tratamiento de las señales mediante hardware diseñado específicamente, y descargar la parte computacional en un soft-processor, o se puede realizar el tratamiento de las señales directamente en el soft-processor. Es por ello, que en este proyecto nos centraremos en obtener la mejor configuración posible de soft-processor, basándonos en FPGAs que proporciona la empresa Xilinx, concretamente en el soft-processor MicroBlaze.

Por último, se ha de destacar que consiguiendo un sistema empuado genérico y potente se optimizarían en gran medida los costes de desarrollo de estos sistemas.

1.2 Objetivos del proyecto

El objetivo fundamental del proyecto es dar solución a los problemas anteriormente descritos. Dicho objetivo principal se puede dividir en dos objetivos principales:

- **Diseñar y desarrollar un procesador empotrado** genérico basado en la tecnología FPGA que tenga como meta conseguir el mejor compromiso entre el rendimiento computacional y el consumo eléctrico.
- **Diseñar y desarrollar benchmarks** o pruebas de rendimiento, que sean capaces de evaluar la capacidad computacional de un procesador.

Tomando como base estos dos objetivos principales, se proponen los siguientes objetivos concretos:

- Evaluar mediante benchmarks la capacidad computacional del juego de instrucciones de la arquitectura MicroBlaze.
- Evaluar mediante benchmarks la capacidad computacional en coma flotante de la arquitectura MicroBlaze.
- Evaluar mediante benchmarks la capacidad de las transferencias de memoria en la arquitectura MicroBlaze.
- Evaluar los consumos eléctricos de una FPGA para distintas configuraciones.
- Evaluar la capacidad de ampliación de la arquitectura MicroBlaze, bien mediante dispositivos conectados al bus del MicroBlaze, o mediante coprocesadores añadidos al MicroBlaze.
- Migrar los benchmarks desarrollados a otros procesadores comerciales, y comparar los resultados obtenidos con dichas arquitecturas.
- Comparar la eficiencia energética de la arquitectura MicroBlaze respecto de las otras arquitecturas.

1.3 Contenido de la memoria

En este apartado se explica el contenido de este documento, detallando el contenido de cada uno de los capítulos que nos podemos encontrar. Ahora se va a proceder a explicar brevemente los principales capítulos que componen este documento:

1. **Introducción y objetivos:** En este capítulo se muestra la motivación de este proyecto, así como los objetivos que se esperan conseguir.
2. **Estado del arte:** Este capítulo muestra la visión de las tecnologías y aplicaciones usadas, así como la evolución tecnológica que ha permitido el desarrollo del sistema. En las diferentes secciones se explicará cada concepto utilizado en este proyecto.
3. **Análisis:** En este capítulo se muestran los conjuntos de requisitos y especificaciones concretas que son necesarias para poder llevar a cabo el proyecto.
4. **Diseño:** Agrupa todas las soluciones propuestas a las anteriores especificaciones, así como la forma de actuar para el desarrollo del proyecto.
5. **Implementación:** En este capítulo se explica el proceso seguido para la integración de las plataformas hardware y software.
6. **Evaluación y análisis de resultados:** Esta es la sección más importante del documento, en ella se evalúa el software desarrollado, y se analizan las distintas alternativas de sistemas empujados realizados.
7. **Planificación y presupuesto:** Lista detallada de las tareas y tiempos que se han tardado en realizar, y una lista detallada de los costes que supone desarrollar, implantar y mantener el sistema.
8. **Conclusiones y trabajos futuros:** En este apartado se analizan los problemas más importantes que se han encontrado en el desarrollo del proyecto, y las conclusiones obtenidas en el desarrollo del mismo. También tiene cabida las futuras líneas de investigación del proyecto.
9. **Acrónimos y abreviaturas:** Se describen cada uno de los acrónimos y abreviaturas mencionadas en los distintos capítulos del documento.
10. **Bibliografía:** En este capítulo se ilustra todo el material consultado a la hora de realizar el presente documento.



Además se incluyen una serie de anexos para la correcta utilización y comprensión del sistema:

1. **Anexo A – Material entregado:** listado de todos los benchmarks y configuraciones de FPGA entregados.
2. **Anexo B – Resumen de arquitecturas implementadas:** listado de todos los sistemas empotrados basados en FPGAs creados para el proyecto, y resumen de todas sus características.

2 Estado del Arte

En este capítulo se presenta un estudio de las diferentes tecnologías existentes: evaluadores de rendimiento, sistemas empotrados y arquitecturas de procesadores. De esta manera podemos entender y comprender qué es lo más recomendable en términos tecnológicos para el proyecto.

Además, también vamos a analizar productos similares que existen ya en el mercado, observando exhaustivamente sus características y puntos clave.

2.1 FPGAs

Las FPGAs fueron inventadas en el año 1984 por Ross Freeman y Bernard Vonderschmitt, co-fundadores de Xilinx. Estas surgieron como una evolución de los CPLDs y los PAL. Concretamente, una FPGA es un dispositivo semiconductor que contiene bloques de lógica cuya interconexión y funcionalidad puede ser configurada ‘in situ’ mediante un lenguaje de descripción especializado.

Esta lógica programable puede reproducir desde funciones sencillas llevadas a cabo por una puerta lógica, hasta sistemas combinacionales complejos e incluso sistemas ‘system-on-a-chip’, o lo que es lo mismo, ordenadores completos incluidos en un único circuito integrado.

Las características principales de las FPGAs son las siguientes^{[1][2]}:

- Gran capacidad de procesamiento de información.
- Paralelismo, dado que se pueden implementar varios procesos concurrentemente.
- Permiten probar diseños antes de realizar una implementación en dispositivos reales.
- Reducen los costes e incrementan el valor de los sistemas producidos.
- Permiten un prototipado de dispositivos rápido.
- Permiten crear familias de productos de manera versátil.

Viendo qué nos ofrecen las FPGAs, podemos concluir que pueden llegar a ser dispositivos muy potentes y competentes. Ahora vamos a ver qué empresas proporcionan soluciones para sistemas empujados basados en FPGAs.

2.1.1 Xilinx

Xilinx, Inc. fue fundada en 1984 por dos ingenieros, Ross Freeman y Bernard Vonderschmitt, los cuales trabajaron en circuitos integrados y dispositivos de estado sólido en Zilog Corporation. La sede está en San José, California, y dispone de oficinas adicionales en distintas partes del mundo^[5]. En la actualidad su objetivo de mercado está centrado en la venta de dispositivos FPGA, CPLD, herramientas de diseño, diseños base de dispositivos y los denominados “IP core”.

Centrándonos en las FPGAs, Xilinx dispone de una gran variedad de soluciones, incluyendo desde modelos de alto rendimiento (serie Virtex) a modelos de bajo coste (serie Spartan). Los modelos y características principales de Xilinx son los siguientes^[3]:

Features	Artix-7	Kintex-7	Virtex-7	Spartan-6
Logic Cells	360,000	480,000	2,000,000	150,000
BlockRAM	19Mb	34Mb	68Mb	4.8Mb
DSP Slices	1,040	1,920	3,600	180
DSP Performance (symmetric FIR)	1,306GMACs	2,845GMACs	5,335GMACs	140GMACs
Transceiver Count	16	32	96	8
Transceiver Speed	6.6Gb/s	12.5Gb/s	28.05Gb/s	3.2Gb/s
Total Transceiver Bandwidth (full duplex)	211Gb/s	800Gb/s	2,784Gb/s	50Gb/s
Memory Interface (DDR3)	1,066Mb/s	1,866Mb/s	1,866Mb/s	800Mb/s
PCI Express® Interface	x4 Gen2	x8 Gen2	x8 Gen3	x1 Gen1
Agile Mixed Signal (AMS)/XADC	Yes	Yes	Yes	
Configuration AES	Yes	Yes	Yes	Yes
I/O Pins	600	500	1,200	576

Tabla 1: Modelos principales de las FPGA de Xilinx

Dentro de cada modelo, existen distintas variantes, incluyendo modelos de gama baja, media y alta por cada serie. En cuanto a las herramientas de desarrollo, dispone de dos grandes entornos de desarrollo^[4]:

- **ISE Design Suite:** Es la herramienta principal de sintetizado y análisis de diseños HDL, permitiendo el desarrollo de diseños en las FPGA. Además permite realizar análisis de tiempos mediante simulación, examinar diagramas RTL y aplicar los diseños configurados a las FPGA. Incluye el sistema MicroBlaze, con el que se permite la creación de sistemas empotrados. Entre todas las aplicaciones que se incluyen, se podrían destacar las siguientes:
 - Platform Studio: Herramienta principal de creación de soft-processor.
 - Software Development Kit: Herramienta de desarrollo de software para soft-processor.
 - MicroBlaze Soft Processor: El procesador software de Xilinx.
 - MicroBlaze Microcontroller System: Conjunto de cores para el funcionamiento de MicroBlaze.
- **Vivado Design Suite:** Es la evolución del ISE Design Suite, diseñado para los nuevos modelos de FPGAs de la serie 7 (Artix, Kintex y Virtex) y la plataforma de procesamiento Zynq 7000, además de las futuras generaciones de FPGAs de Xilinx. Consta de 16 herramientas individuales, incluyendo un IDE, la librería completa de núcleos IP de Xilinx, y herramientas de sintetizado que acepta lenguajes como C, C++ y SystemC. Está diseñado principalmente para la generación de sistemas ASIC y “System-on-a-chip” tanto de procesamiento sencillo (SoC) como múltiple (MPSoC). Las aplicaciones más importantes que se incluyen son las siguientes:
 - Vivado IDE.
 - Software Development Kit (SDK).
 - Vivado Integrated Design Environment for MicroBlaze.

Viendo qué herramientas de desarrollo y diseño nos ofrece Xilinx, en caso de utilizar su tecnología, resultaría la más adecuada la serie de FPGAs Spartan 6 y el entorno de trabajo IDE Design Suite. De esta manera podríamos diseñar sistemas empotrados basados en FPGAs con la arquitectura MicroBlaze, y a un bajo coste (las licencias son gratuitas, mientras que el precio de la plataforma también es reducido).

2.1.2 Altera

Altera Corporation fue fundada en 1983, consiguiendo su primer dispositivo programable por hardware en 1984. En la actualidad su estrategia de mercado es la venta de dispositivos FPGAs, CPLDs, ASICs y herramientas de diseño. En cuanto a las FPGAs, Altera ofrece las siguientes series^[6]:

- **Serie Cyclone V:** Serie de bajo coste y bajo consumo. Están especializadas para el uso en el sector industrial, móviles, radiodifusión por cable y mercados de consumo. Hay una gran cantidad de IP cores ya existentes para esta serie, por lo que resulta fácil y sencillo realizar diseños y trasladarlos a dispositivos reales. Esta serie se creó en el 2011, está realizada en un proceso de fabricación de 28 nanómetros, y añade la posibilidad de agregar un procesador adicional basado en la arquitectura ARM.
- **Serie Arria V:** Es la serie orientada a la gama media, siendo ésta optimizada en el bajo consumo. Están especializadas en dispositivos de radio, dispositivos de línea telefónica y equipamiento de estudio para redifusión. Al igual que la serie Cyclone, cuenta con gran cantidad de IP cores, lo que permite maximizar sus capacidades, a la vez que facilita el diseño y sintetizado. También están fabricadas en 28 nanómetros, y tienen la posibilidad de agregar un procesador basado en la arquitectura ARM.
- **Serie Stratix V:** Dedicada a ofrecer el mayor ancho de banda y el mayor nivel de integración a nivel de sistema, esta serie es la especializada en sistemas de alto nivel donde es necesario la mayor potencia posible. Sus aplicaciones pueden ser como servidor de video para estudios, tarjetas de tipo Channel y RF, radares militares, *switches* o tarjetas de red de tipo óptica o Gigabit Ethernet. Por otro lado, cabe destacar que esta serie carece de la posibilidad de incluir un procesador de tipo ARM.

Por otro lado, Altera posee diversas herramientas de desarrollo. Las más importantes son las siguientes^[7]:

- **Quartus II:** Es la suite de diseño, permite el análisis y sintetizado de diseños HDL, realizar análisis de tiempos, examinar diseños RTL, simular diseños completos y programar los diseños realizados en las FPGAs. Existen dos versiones, una gratuita denominada Web Edition, y otra de pago denominada Subscription Edition. La diferencia principal entre ambas reside en el soporte de los modelos y series de Altera, mientras que la versión gratuita solo permite CPLDs y la serie Cyclone, la versión de pago permite obtener toda la funcionalidad y dispositivos de Altera.

- **Nios II Embedded Design Suite (EDS):** Es una colección de herramientas, utilidades, librerías y drivers para el desarrollo de sistemas empujados, basados en la arquitectura Nios II de Altera. Está totalmente basado en Eclipse, de tal manera que se facilita la inclusión de plugins con los que incrementar la funcionalidad y la productividad. Estas son las principales características del EDS:
 - Navegación y edición de código fuente.
 - Debugging y profiling de código fuente.
 - Compilado y enlazado de código escrito en C y C++.
 - Plugins para el organizador de proyectos, diseños base, programación directa de FPGAs, edición de BSPs y línea de comandos.

Una vez hemos visto qué herramientas de diseño, y qué series de FPGAs dispone Altera, procedemos a seleccionar la serie Cyclone, junto con las herramientas de desarrollo Quartus II y Nios II Embedded Design Suite. Así podríamos realizar diseños de sistemas empujados, basados en la arquitectura Nios II implementada en una FPGA.

Ahora vamos a analizar en profundidad qué arquitectura de soft-processor resulta más conveniente utilizar, repasando primeramente los conceptos básicos y la historia de los mismos.

2.2 Soft-Processors

Los soft-processors son procesadores que pueden ser implementados en una FPGA, CPLD o ASIC, a través de la síntesis en puertas lógicas. Cabe destacar que es posible utilizar más de un soft-processor dentro de un mismo dispositivo con lógica programable, pero por la naturaleza de los mismos, resulta complicada la compartición de recursos hardware.

Por otro lado, dependiendo del tamaño de la FPGA, es posible implementar procesadores más complejos o con más recursos. A continuación se van a enumerar distintos tipos de soft-processors existentes con sus características más importantes.

2.2.1 Arquitectura MicroBlaze

La arquitectura MicroBlaze es la propuesta por Xilinx, y puede ser implementada en prácticamente cualquier modelo de FPGA de esta marca. Su diseño interno está basado en la tecnología RISC, siendo muy parecido a la arquitectura DLX, diseñada por John L. Hennessy y David A. Patterson (diseñadores principales de las arquitecturas MIPS y RISC). El diseño de esta arquitectura surgió en el año 2001, y obtuvo soporte por primera vez en la versión 3.1 del Xilinx EDK.

La arquitectura MicroBlaze es capaz de ejecutar la mayor parte de las instrucciones en un único ciclo, manteniendo de esta manera un rendimiento elevado. Las características principales de esta arquitectura son las siguientes^[8]:

- Soporte de Big-Endian y Little-Endian.
- Bus de memoria LMB, propio del MicroBlaze con hasta 64Kb de memoria.
- Registros de uso general de 32 bits.
- Unidad de memoria virtual (opcional y configurable).
- Soporte de memoria caché, en modo write-through o write-back (opcional y configurable).
- Modos stream y víctima configurable en la caché.
- Bus FSL (Fast Simple Link) para el uso de coprocesadores.
- Bus de dispositivos PLB o AXI.
- Pipeline de 3 o 5 etapas (optimización de área u optimización de rendimiento).
- Unidad de coma flotante de precisión simple (opcional y configurable).
- Instrucciones de división, multiplicación de 64 bits, comparación y shifting de bits aceleradas por hardware (opcional y configurable).
- Caché de predicción de saltos (opcional y configurable).
- Delayed slot para saltos.
- Control de errores (opcional).
- Soporta los sistemas operativos XilKernel y Linux.
- Compilador GCC, alta compatibilidad y optimización.

El diseño interno del procesador MicroBlaze se puede ver en la Ilustración 1:

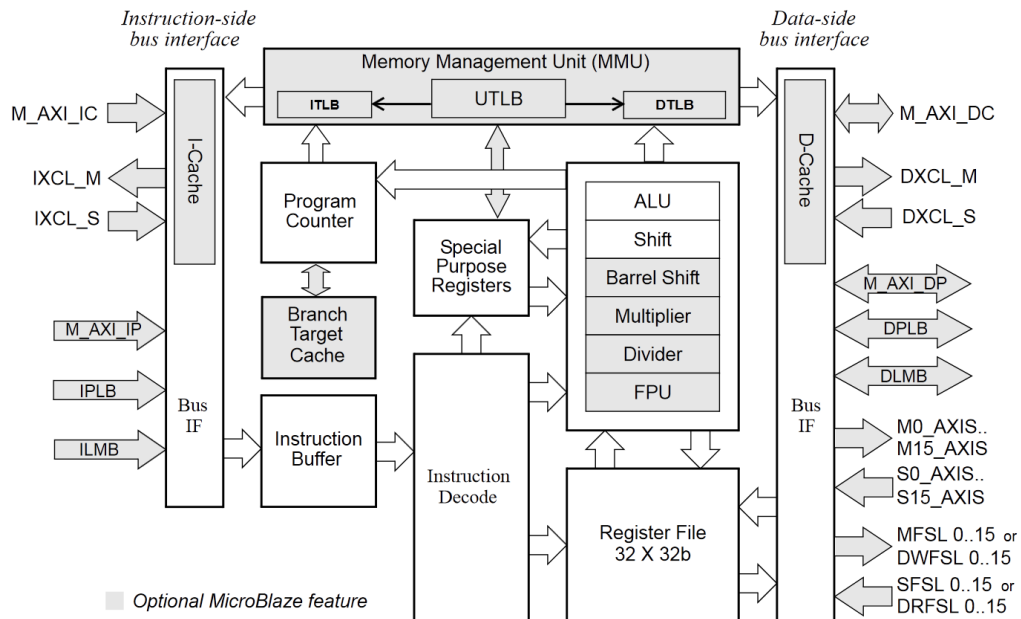


Ilustración 1: Arquitectura del procesador MicroBlaze

Tal y como podemos observar, la arquitectura MicroBlaze puede ser considerada como una buena opción, puesto que resulta altamente configurable, además de facilitar su implementación a través de las herramientas de diseño y desarrollo de Xilinx.

2.2.2 Arquitectura Nios II

La arquitectura Nios II es la propuesta por Altera, se trata de un procesador de 32 bits diseñado para las FPGAs de Altera. Es la evolución directa de la arquitectura Nios de 16 bits, haciéndola más potente y más adaptable a todo tipo de dispositivos, incluyendo desde procesamiento digital hasta control de sistemas.

Su diseño interno está también basado en la tecnología RISC, y cuenta con distintos modelos dependiendo de las necesidades. Todos comparten un espacio de memoria de 32 bits, una unidad de coma flotante en precisión simple y la opción de utilizar el acelerador Nios II C-to-Hardware Acceleration Compiler, que traduce código en C a hardware para obtener un mejor rendimiento.

Estos son los modelos y las características que disponen^[9]:

- **Nios II/f:** Es el modelo diseñado para ofrecer un máximo rendimiento, a expensas del tamaño ocupado. Las características que ofrece son las siguientes:
 - Caché de datos e instrucciones separadas, desde 512 bytes hasta 64 Kb.
 - Unidad de memoria virtual (opcional).
 - Espacio de memoria de hasta 2 Gb.
 - Memoria opcional de alto rendimiento para instrucciones y datos.
 - Pipeline de 6 etapas.
 - Multiplicación y shifting de bits aceleradas por hardware (1 ciclo de instrucción).
 - División acelerada por hardware (opcional).
 - Predicción de saltos dinámica.
 - Hasta 256 instrucciones agregables, y posibilidad de usar aceleradores hardware.
 - Modulo de debugging a través de JTAG.

- **Nios II/s:** Modelo estándar, ofrece buen rendimiento sin sacrificar el tamaño ocupado en la FPGA. Las características que ofrece son las siguientes:
 - Caché de instrucciones.
 - Memoria opcional de alto rendimiento para instrucciones.
 - Pipeline de 5 etapas.
 - Predicción de saltos estática.
 - Multiplicación y shifting de bits aceleradas por hardware (1 ciclo de instrucción).
 - División acelerada por hardware (opcional).
 - Hasta 256 instrucciones agregables.
 - Modulo de debugging a través de JTAG.

- **Nios II/e:** Modelo que intenta utilizar la menor cantidad de lógica para su implementación en una FPGA. Diseñado específicamente para los modelos de bajo coste Cyclone II. Las características que ofrece son las siguientes:
 - Modulo de debugging a través de JTAG.
 - Ocupa menos de 700 LEs.
 - Hasta 256 instrucciones agregables.

El diseño interno del procesador Nios II se puede ver en la Ilustración 2:

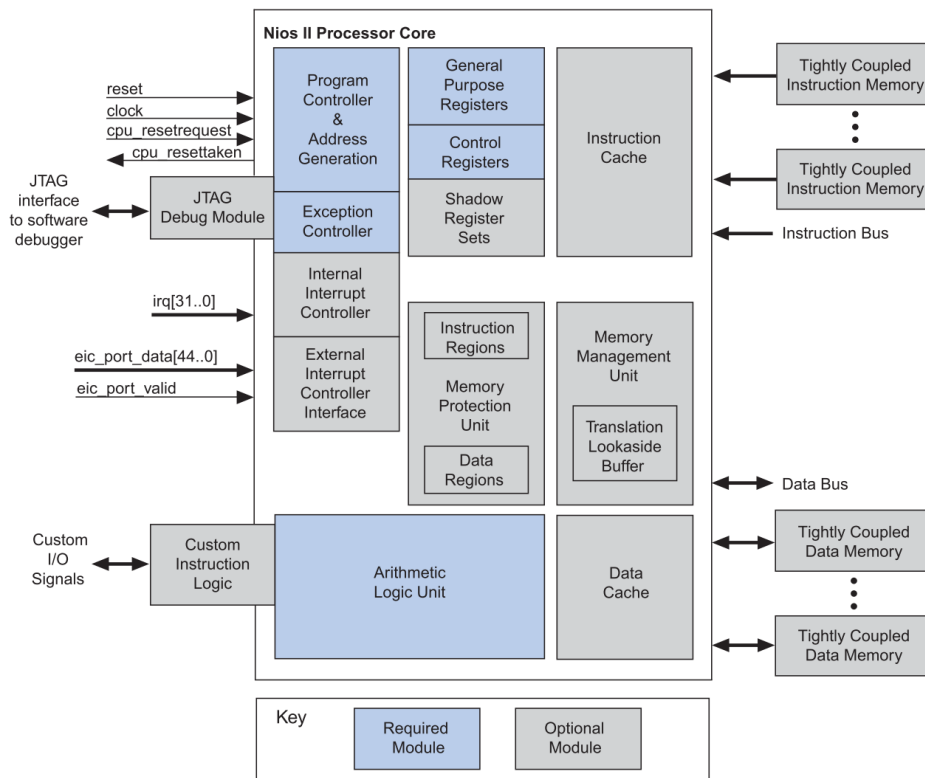


Ilustración 2: Arquitectura del procesador Nios II

La arquitectura Nios II parece también una buena opción, aunque resulta menos adaptable que la arquitectura MicroBlaze. Ahora vamos a ver dos alternativas totalmente libres de soft-processors.

2.2.3 Arquitectura LEON 4

La arquitectura LEON 4 se deriva del proyecto LEON. Este proyecto fue iniciado por la Agencia Espacial Europea (ESA) a finales de 1997, con el objetivo de estudiar y desarrollar un procesador de alto rendimiento para su uso en los proyectos espaciales. El objetivo por aquel entonces fue conseguir un diseño abierto, portable y sin licencias propietarias, capaz de proporcionar un alto rendimiento, alta compatibilidad y bajo coste. El proyecto más adelante pasó a pertenecer a Aeroflex Gaisler, los cuales desarrollaron las arquitecturas LEON 3 y LEON 4.

La arquitectura LEON 4 cumple la especificación SPARC-V8 RISC y fue lanzada en Enero de 2010, y centra todo su estructura de software en el GRLIB, que es un conjunto de IP cores totalmente funcional. Al igual que los procesadores de gama alta, su diseño contempla el uso de una caché L1, y una caché opcional de tipo L2.

El diseño interno del procesador Leon4 se puede ver en la Ilustración 3:

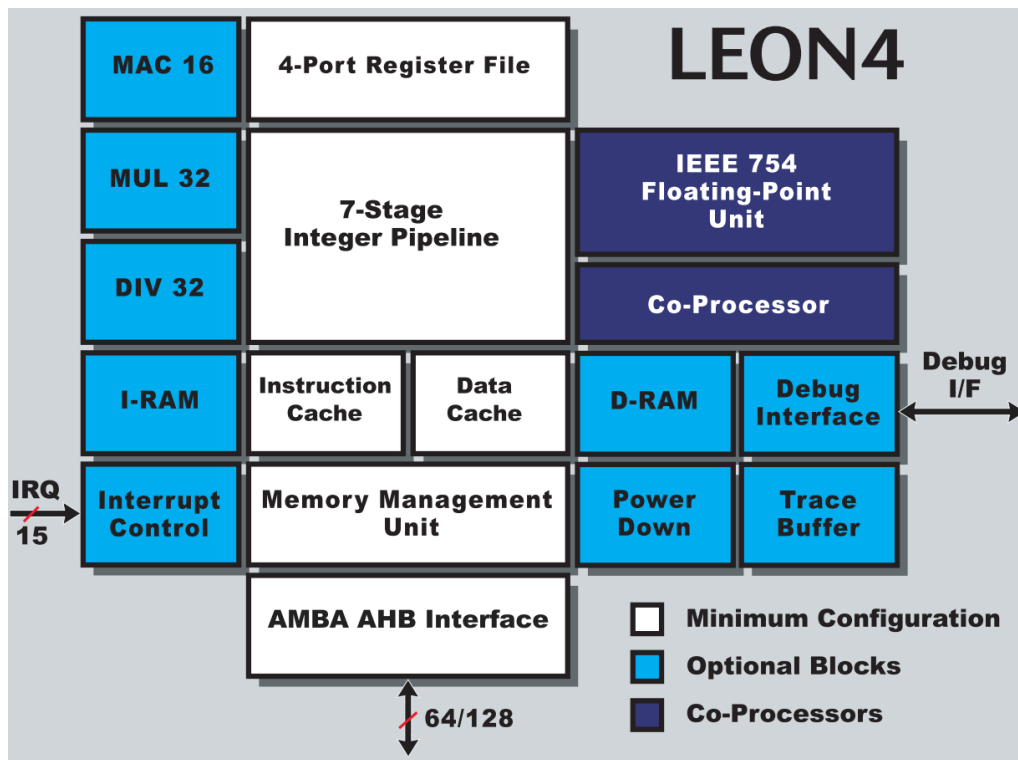


Ilustración 3: Arquitectura del procesador LEON 4

Las características principales disponibles son las siguientes^[10]:

- Juego de instrucciones SPARC V8 con la extensión V8e (big-endian).
- Pipeline de 7 etapas, con predicción de saltos.
- Operaciones de lectura/escritura de un solo ciclo en 64 bits.
- Unidades MAC, multiplicación y división realizadas en hardware.
- Unidad de coma flotante con normativa IEEE-754, totalmente desarrollada en el pipeline.
- Caché de datos e instrucciones separadas en el nivel L1, con snooping.
- Caché L1 configurable, de 1 a 4 vías, y de 1 Kb a 256 Kb por vía. Estrategia de reemplazo seleccionable entre aleatoria, LRR o LRU.
- Caché L2 configurable, de 1 a 4 vías y tamaño entre 16 Kb y 8 Mb.
- Unidad de memoria virtual SPARC, con TLB configurable.
- Bus de datos AMBA 2.0 AHB, de 64 o 128 bits.
- Debugging integrado, con buffer de traza para instrucciones y datos, medidor de rendimiento.
- Soporta multiprocesador (SMP).
- Reducción de consumo dependiendo del consumo, y capaz de cambiar la frecuencia del mismo.
- Diseño optimizado para la ejecución de instrucciones en un solo ciclo.
- Sintetizable en cualquier FPGA, incluyendo aquellas de Xilinx y Altera.
- Soporte nativo de Linux.

Como podemos observar, esta arquitectura parece ser realmente potente, ofreciendo características muy interesantes respecto a las arquitecturas vistas anteriormente. Sin embargo, el empleo de GRLIB dificulta en cierta medida la implementación de esta arquitectura, puesto que no todas las FPGAs están contempladas.

Un ejemplo de esta deficiencia, es la ausencia de soporte para algunas placas actuales, como puedan ser la Digilent Atlys o la Digilent Nexys3 (modelos reconocidos en los entornos universitarios). Además, solamente se proveen los ficheros VHDL, por lo que no resulta fácil la tarea de implementación dentro de las FPGAs, al estar ausentes los ficheros de mapeo para las placas.

2.2.4 Arquitectura OpenRISC 1000

La arquitectura OpenRISC es el proyecto principal de la comunidad de desarrolladores OpenCores, y su objetivo es el desarrollo de una serie de arquitecturas RISC de código totalmente abierto. El único diseño finalizado y validado hasta la fecha es el OpenRISC 1000, describiendo una familia de procesadores de 32 y 64 bits, con unidades de coma flotante y unidades vectoriales opcionales.

La primera implementación llevada a cabo del OpenRISC 1000 es el OpenRISC 1200, escrita totalmente en lenguaje Verilog. La licencia bajo la que ha sido desarrollado es la LGPL para el diseño hardware, con los modelos y el firmware lanzados bajo la licencia GPL. A partir de esta arquitectura, también se ha lanzado una implementación SoC conocida como ORPSoC.

El diseño interno del procesador OpenRISC 1200 se puede ver en la Ilustración 4:

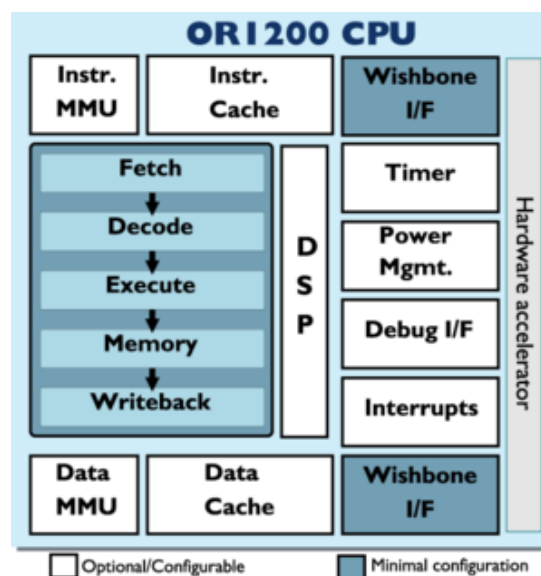


Ilustración 4: Arquitectura del procesador OpenRISC 1200

En cuanto a las características principales, las más importantes son las siguientes^[11]:

- Unidad de CPU/DSP central.
- Unidad de coma flotante de precisión simple IEEE-754.
- Caché de datos mapeada directamente.
- Caché de instrucciones mapeada directamente.
- Unidad de memoria virtual basada en tablas hash DTLB.
- Unidades de administración de consumo, e interfaz de control de dicha administración.
- Timer de ticks.
- Unidades para el desarrollo y debugging.
- Controlador de interrupciones.
- Bus de datos Wishbone.

Además de estas características, cabe destacar que gracias a su implementación en el lenguaje Verilog, éste resulta sintetizable en gran cantidad de modelos de FPGAs incluyendo las de Xilinx y Altera. Por otro lado, están completamente soportadas las herramientas de compilación GCC y LLVM, y soporta sistemas operativos como Linux.

Al igual que la arquitectura LEON 4, existen diversos modelos de FPGAs que no tienen todavía implementación alguna de esta arquitectura, por lo que queda en cierto sentido limitada. Además, al ser un diseño realizado por una comunidad de desarrolladores, existen diversos aspectos en los que la arquitectura debería ser mejorada, puesto que el rendimiento obtenido por los usuarios no es el esperado.

2.3 Arquitecturas principales

Con distintas arquitecturas de soft-processors vistas, ahora vamos a analizar brevemente las distintas arquitecturas ya existentes, con las que podremos comparar en rendimiento al soft-processor que elijamos. Nos especializaremos en la especificación de estas arquitecturas para sistemas empujados, dado que dispondrán de una potencia similar a los soft-processors que utilicemos en este proyecto.

2.3.1 Arquitectura x86

La arquitectura x86 fue introducida en el año 1976, como expansión de 16 bits del procesador 8080 de Intel. Su diseño está basado en la tecnología CISC, pero con el paso del tiempo su implementación interna ha acabado siendo de tipo RISC.

Hoy en día este juego de instrucciones ha sido ampliado, llegando a los 32 bits (IA-32, con la introducción del procesador Intel 80386) y 64 bits (x86-64, con la introducción del procesador AMD Athlon 64). También ha sido ampliado con instrucciones de tipo SIMD, con los juegos de instrucciones MMX y SSE (y otros poco utilizados como 3DNow de AMD). La base de IA-32 es la que ha prevalecido, y la que hoy en día se implementa en todos los procesadores x86^[12].

Los mercados a los que ha sido adaptado x86 incluyen desde los supercomputadores, hasta los sistemas empotrados o sistemas de bajo consumo (por ejemplo, dispositivos móviles o tabletas). Puesto que nos vamos a centrar en los sistemas empotrados, vamos a nombrar algunas de las implementaciones de x86 en este tipo de mercado, aunque no sea una arquitectura completamente adecuada para este cometido. El problema reside en que el diseño CISC contempla un conjunto de instrucciones muy amplio y complejo, lo que ocasiona que el consumo energético sea mayor, al estar dificultada la reducción en tamaño de las arquitecturas. Algunos ejemplos de estos procesadores para entornos industriales son los siguientes:

- **Intel 386EX:** Edición para sistemas empotrados del procesador Intel 80386SX, cuenta con un bajo consumo y con compatibilidad completa con la arquitectura IA-32.
- **Intel 486GX:** Es una edición especial de los procesadores Intel 80486, orientados al bajo consumo, y a su aplicación en diseños industriales. Este modelo duro poco en el mercado, y su éxito fue más bien reducido.
- **DM&P Electronics Vortex86:** Es un núcleo compatible con la arquitectura x86, e implementada en sistemas SoC. La implementación original de este procesador viene de la realizada por Rise Technology en sus procesadores mP6.
- **ZF Micro ZFx86:** Basado en los procesadores Cyrix Cx486DX, constituye un SoC completo y de bajo consumo (menos de 1 Vatio a 100 MHz). Está específicamente diseñado para entornos industrializados, por lo que cuenta con medidas avanzadas de seguridad (ROM de arranque y de sistema empotrado en el propio procesador).
- **RDC Semiconductors R8610 y R8620:** Procesadores empotrados, compatibles con el juego de instrucciones de los procesadores Intel 486SX. Alcanzan frecuencias de unos 150Mhz, con un consumo de 1 Vatio.

Como se puede observar, todos estos procesadores empotrados con tecnología x86 utilizan tecnología basada en la arquitectura del procesador 80486 de Intel o similar, por lo que es de suponer que no tienen un alto rendimiento respecto a los ordenadores actuales de sobremesa, pero si una optimización energética elevada y un rendimiento similar o superior a los alcanzados en FPGAs. Esta optimización energética se basa en la eliminación de las unidades de memoria virtual (no todas) y la eliminación de las unidades de coma flotante.

Hoy en día este campo se esta retomando para los procesadores x86, con Intel y sus procesadores Atom series Z2xxx para los mercados de móviles, y AMD con sus series Embedded G-series y R-series, ambas de mayor potencia y relativamente bajo consumo.

Los procesadores Intel Atom Z2xxx, cuentan con todo lo necesario para su funcionamiento en teléfonos móviles y tabletas^[56]. Sus frecuencias de reloj oscilan entre 1 GHz y 2 GHz, con posibilidad de utilizar uno o dos núcleos, y la capacidad de añadir multiproceso por cada núcleo. Además integran controladores de memoria, dispositivos aceleradores de vídeo y cachés L1 y L2 para instrucciones y datos, lo que los hacen mucho más potentes que los que se utilizan en entornos industriales.

2.3.2 Arquitectura MIPS

MIPS es un acrónimo de “Microprocessor without Interlocked Pipeline Stages”, o lo que es lo mismo, procesador con etapas del pipeline independientes. La arquitectura MIPS está diseñada bajo la tecnología RISC, y es desarrollada por MIPS Computer Systems.

La arquitectura MIPS consta de distintos tipos de juegos de instrucciones, los cuales han ido evolucionando con el paso del tiempo. Existen implementaciones de 32 y 64 bits, y se han desarrollado juegos de instrucciones de tipo SIMD como MIPS-3D o MDMX^[20].

El uso de esta arquitectura esta focalizada en el uso de sistemas empotrados, como routers, video consolas (toda la gama de consolas de Sony hasta 2006), impresoras digitales o PDAs (anteriormente, hasta el año 2006 fueron también utilizados en distintas gamas de Workstation de SGI). Los procesadores empotrados utilizan los juegos MIPS32 4K para 32 bits^[21], y MIPS64 5K para 64 bits^[22], y la reciente evolución MIPS32 M14K diseñada para microcontroladores.

En la actualidad, MIPS ofrece tres tipos de microprocesadores, cada uno de ellos orientado a una gama de productos distintos:

- **microAptiv**: procesador empotrado con arquitectura MIPS32R3, de alta eficiencia energética y pensado para sistemas en tiempo real. Cuentan con unidades DSP y SIMD para el procesamiento de señales. Su uso está orientado al segmento de industrial, medidores de valores, automoción y el sector de las telecomunicaciones.
- **interAptiv**: procesador empotrado con arquitectura MIPS32R3, de diseño multiprocesador. Cuenta con un pipeline de 9 etapas multihilo, otorgando una alta eficiencia y rendimiento. Es ideal para aplicaciones que requieren el uso de cómputos en paralelo, como puedan ser routers, procesamiento de comunicaciones telefónicas, controladores SSD y equipamiento automovilístico.
- **proAptiv**: procesador con diseño out-of-order, arquitectura MIPS32R3, multinúcleo y un alto rendimiento. Está pensado para el uso en dispositivos de control en redes de conexión, y para dispositivos de consumo.

En la Ilustración 5 podemos ver la estructura interna del procesador MIPS32 M14K, orientado a dispositivos microcontroladores de alto rendimiento.

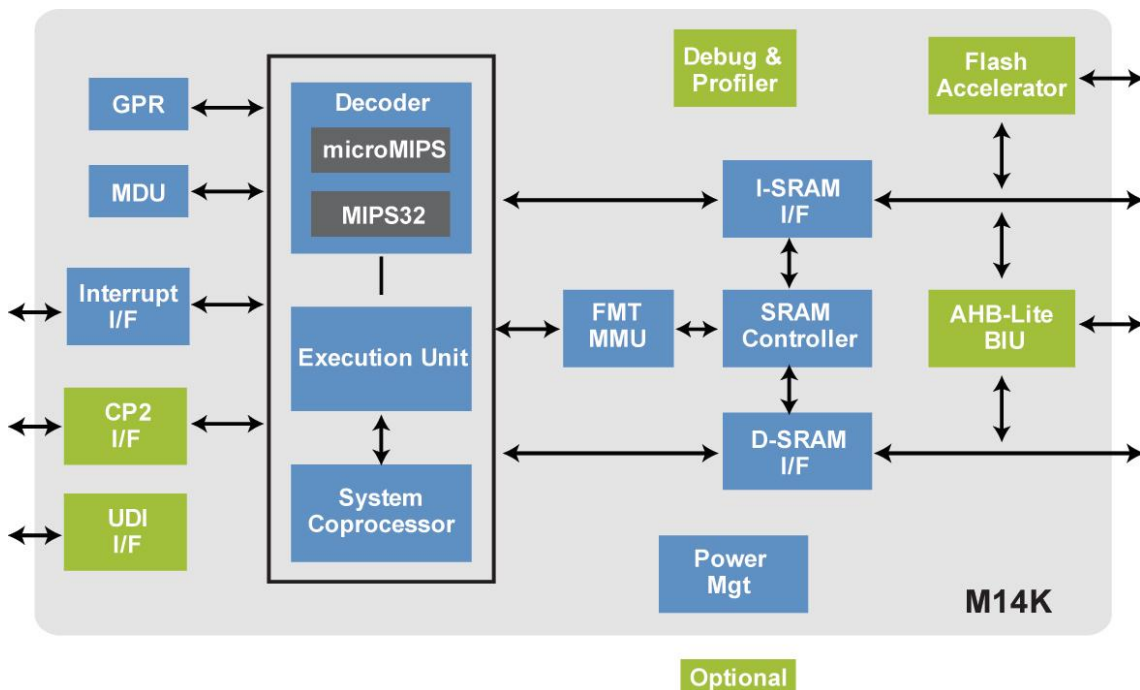


Ilustración 5: Arquitectura del procesador MIPS32 M14K

También han sido desarrollados IP cores basados en la arquitectura MIPS, ampliables con unidades de coma flotante, vectorial, seguridad y otros tipos de

unidades. Altera ha sido de los primeros en realizar dicha tarea, incluyendo el soft-processor MP32 en algunas de sus gamas.

2.3.3 Arquitectura ARM

Finalmente, la arquitectura ARM es, por decirlo de alguna manera, la arquitectura más reconocida en el mundo de los sistemas empotrados, gracias a su buena relación de potencia/consumo. Está basada en la tecnología RISC, y es desarrollada por ARM Holdings. Realmente ARM es un acrónimo de “Advanced RISC Machine”, y fue creada por Acorn Computers para el uso en sus ordenadores personales.

Tomaron como base de la arquitectura el diseño del procesador MOS 6502, diseñándolo para obtener más potencia respecto de éste, con el mismo rendimiento en términos de entrada/salida y con un juego de instrucciones de 32 bits, resultando el primer juego de instrucciones denominado ARMv2. Más adelante, se mejoró la arquitectura, añadiendo cachés de instrucciones y datos, y unidades de memoria virtual. También se añadieron unidades de coma flotante, procesadores DSP, y unidades SIMD como la VFP, o la más actual NEON^[25].

Hoy en día, ARM ha centrado su arquitectura en el mercado de sistemas empotrados, llegando a tener un 90% del mercado de procesadores de 32 bits de tecnología RISC. Es destacable su uso en electrónica de consumo, como móviles, tabletas, videoconsolas, y periféricos como routers o controladoras para equipos informáticos, e incluso se está empezando a plantear el uso de estos procesadores en entornos de supercomputación y servidores a gran escala.

El modo de funcionamiento de ARM es el licenciado del diseño (como ocurre con los IP cores), distribuyendo una descripción hardware del núcleo del procesador adquirido, un completo entorno de desarrollo (compilador, debugger y SDK) y la posibilidad de vender productos con el procesador ARM integrado. Por otro lado, ARM ha especializado la arquitectura para los distintos segmentos del mercado. En la actualidad existen las siguientes series de procesadores ARM^[26]:

- **Series Cortex-A:** Procesadores de alto rendimiento, diseñados para sistemas operativos abiertos. Son multiprocesador de alta frecuencia, con varias unidades de coma flotante y SIMD.
- **Series Cortex-R:** Procesadores exclusivos para aplicaciones con funcionamiento en tiempo real, con bajo consumo y alto rendimiento.
- **Series Cortex-M:** Procesadores dedicados al uso como microcontroladores, tienen un tamaño reducido, y ofrecen el menor consumo posible.
- **Series Classic:** Son las arquitecturas ARM11, ARM9 y ARM7. Ofrecen un precio reducido.

- **Serie especiales:** Son las arquitecturas SecurCore, que contienen módulos específicos de seguridad, y las arquitecturas para FPGAs, que contienen diseños específicos para su implementación en FPGAs como IP cores. El primero de la serie de procesadores específicos para FPGAs es el ARM Cortex-M1, y puede implementarse en FPGAs de Altera, Actel y Xilinx.

En la Ilustración 6 podemos observar la arquitectura ARM11, muy utilizada en dispositivos móviles:

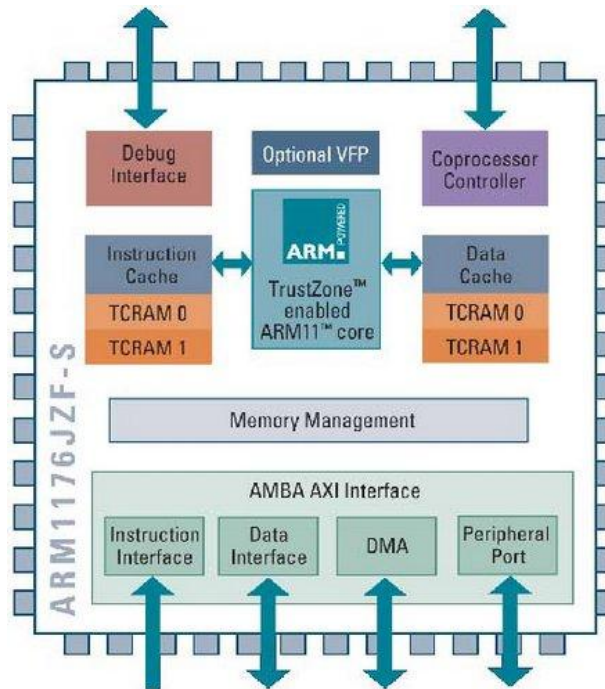


Ilustración 6: Arquitectura de un procesador ARM11

2.4 Pruebas de rendimiento

Las pruebas de rendimiento, o *benchmarks* en inglés, son una técnica para evaluar el rendimiento de un sistema o de un componente del mismo. Existen básicamente dos tipos de benchmarks dentro de los entornos informáticos, los benchmarks sintéticos son programas especialmente creados para comprobar una carga de trabajo determinada, centrada en algún aspecto específico, mientras que los benchmarks de aplicaciones evalúan el rendimiento de aplicaciones ya existentes, y de uso cotidiano.

Generalmente, los benchmarks de aplicaciones demuestran el rendimiento que en la realidad tendría el sistema, mientras que los sintéticos no. Sin embargo, evaluar aspectos específicos de un sistema sólo se puede realizar mediante los benchmarks sintéticos.

En cuanto a qué evaluar con los benchmarks, es posible evaluar desde la potencia de cómputo de un sistema empujado, hasta el ancho de banda disponible para las comunicaciones de entrada/salida. Los tipos de benchmarks más conocidos en la actualidad son algunos de los siguientes:

- **Procesadores y microcontroladores:** Estos benchmarks miden el rendimiento de las unidades de procesador de los sistemas, los más conocidos son los siguientes:
 - **Dhrystone:** Es un benchmark sintético, creado en 1984 por Reinhold P. Weicker, centrado en la evaluación del rendimiento de la ALU de un procesador. Evalúa las llamadas a procedimientos, punteros indirectos, asignaciones y otros elementos. En 1988 se mejoró este benchmark, para evitar que los compiladores eliminasen en las optimizaciones parte del código (que obviamente, no hacía realmente nada). Mide el rendimiento en DMIPS.
 - **Whetstone:** Es un benchmark sintético, escrito en 1972 para la evaluación del rendimiento de coma flotante del procesador. Mide el rendimiento en MWIPS.
 - **Linpack:** Es una librería que realiza álgebra lineal numérica. Fue escrito por Jack Dongarra, Jim Bunch, Cleve Moler y Gilbert Stewart. Básicamente, su funcionamiento radica en la utilización de sub-programas de cálculo de álgebra lineal básica, realizando operaciones de vectores y matrices. Mide el rendimiento del procesador de coma flotante.
 - **Livermore loops:** Benchmark diseñado para ordenadores multiprocesador, su funcionamiento consiste en la realización de 24 bucles “do while”, algunos de los cuales pueden ser vectorizados y otros no. Cada bucle realiza un determinado kernel, siendo los kernels cálculos matemáticos específicos.
 - **CoreMark:** Desarrollado en 2009, es un intento de la industria junto con la EEMBC por reemplazar el antiguo benchmark Dhrystone. Realiza varias evaluaciones distintas: procesador de listas, operaciones matemáticas de matrices, máquinas de estados y corrección de errores CRC.

- **Memoria:** Miden la capacidad de un sistema de transferir información entre sus distintos elementos. Tienen la problemática de que cada arquitectura funciona distinto, y los lenguajes de programación no incluyen directivas exclusivas que faciliten este tipo de análisis, por lo que se recae en el uso de código en ensamblador. Algunos de ellos son los siguientes:
 - **RAMspeed:** Benchmark de código libre, que mide el rendimiento de las memorias caché y RAM de ordenadores personales. Está diseñado exclusivamente para ordenadores con las arquitecturas i386, amd64 y alpha. Ofrece un total de 18 benchmarks distintos.
 - **BusSpeed y MemSpeed:** Creados por Roy Longbottom, evalúan el rendimiento de memoria RAM y caché, utilizando distintos juegos de instrucciones (básico, MMX o SSE) en ordenadores con arquitectura i386 y amd64.
 - **STREAM:** Está diseñado para su utilización en la industria, y mide el ancho de banda sostenible en MB/s de la memoria del sistema.
- **Comunicaciones:** Miden la capacidad de un sistema de realizar comunicaciones con el exterior, sobre todo mediante protocolos de red.
 - **NetPerf:** Mide el rendimiento de distintos tipos de comunicaciones, evaluando el rendimiento unidireccional, bidireccional, y la latencia punto a punto. Soporta los protocolos TCP y UDP, mediante IPv4 e IPv6.
 - **NetSpec:** Es una herramienta sofisticada que realiza procedimientos experimentales que evalúan el rendimiento del rendimiento en redes. Evalúa una gran cantidad de tipos de escenarios.

En el capítulo 0, veremos cuáles de ellos utilizaremos, y cuáles hemos de implementar. Por otro lado, con las tecnologías que vamos a utilizar ya vistas, podemos seguir con el siguiente capítulo. En él realizaremos el análisis de las necesidades del proyecto, creando una serie de requisitos para el mismo.



3 Análisis

En este apartado se define el análisis de las pruebas de rendimiento y el sistema a desarrollar, basándose este documento en la metodología de desarrollo de la Agencia Espacial Europea (ESA) en su versión Lite^[51]. Algunos de los apartados se han modificado o eliminado, considerando esta adaptación necesaria para el proyecto. Además, la edición Lite resulta conveniente puesto que está pensada para proyectos software de tamaño reducido.

Primeramente se realizará la elección de la plataforma de trabajo. Una vez se haya establecido la plataforma hardware para el proyecto, se procede a definir de forma detallada cada una de las funcionalidades requeridas para poder evaluar el rendimiento de las FPGAs. Así el lector podrá tener una visión aproximada de cómo serán las pruebas de rendimiento finalmente.

De esta manera, quedarán reflejadas las condiciones generales que tiene el proyecto, así como sus capacidades. Por último, se detallarán todos y cada uno de los requisitos de usuario analizados y extraídos a partir de lo expuesto en capítulos anteriores.

3.1 Elección de plataforma de trabajo

En el capítulo anterior vimos las distintas alternativas de soft-processors, FPGAs y entornos de diseño y desarrollo, y ahora vamos a seleccionar la mejor alternativa. Se ha considerado que la alternativa mejor para este proyecto es la solución propuesta por la empresa Xilinx, utilizando el soft-processor que ellos han desarrollado, el MicroBlaze.

La elección de esta plataforma se ha basado en el rendimiento que puede proporcionar el soft-processor: características técnicas, facilidad de desarrollo e integración, y accesibilidad a las placas de desarrollo económicas. La plataforma propuesta por Altera ha sido descartada por la poca personalización que tiene el procesador Nios II (únicamente configurables los tamaños de caché, y la opcionalidad de unidad de memoria virtual y aceleración de división). Los soft-processors LEON 4 y OpenRISC 1200 también han sido descartados por su difícil implementación dentro del hardware, al no ser directa en las FPGAs. Además, la propuesta del procesador MicroBlaze incluye la posibilidad de utilizar dos tipos de kernels distintos, el propietario de Xilinx denominado XilKernel, y Linux.

Utilizaremos dos placas de desarrollo distintas, ambas proporcionadas por la empresa Digilent. Éstas utilizan la serie Spartan 6, una de gama baja dentro de la serie, y otra de gama media/alta^[32]. Las placas utilizadas son las siguientes:

- **Digilent Nexys3:** Esta placa cuenta con la FPGA Xilinx Spartan 6 LX16, siendo el modelo de gama baja dentro de la serie Spartan 6. Esta FPGA cuenta con un total de 14.579 celdas lógicas, un máximo de 136 Kbits de memoria RAM distribuida, 32 unidades DSP48A1 (multiplicador 18x18, un sumador y un acumulador), un máximo de 576 Kbits de memoria RAM de bloque, y un total de 232 puertos de entrada/salida. Además de estas características la placa ofrece 16 MB de memoria PSRAM (SRAM simulada, basada en tecnología DRAM), un puerto UART a través de USB, un puerto HID a través de USB, un conector Ethernet y uno VGA. En la Ilustración 7 se puede ver el diseño de la placa^[30]:

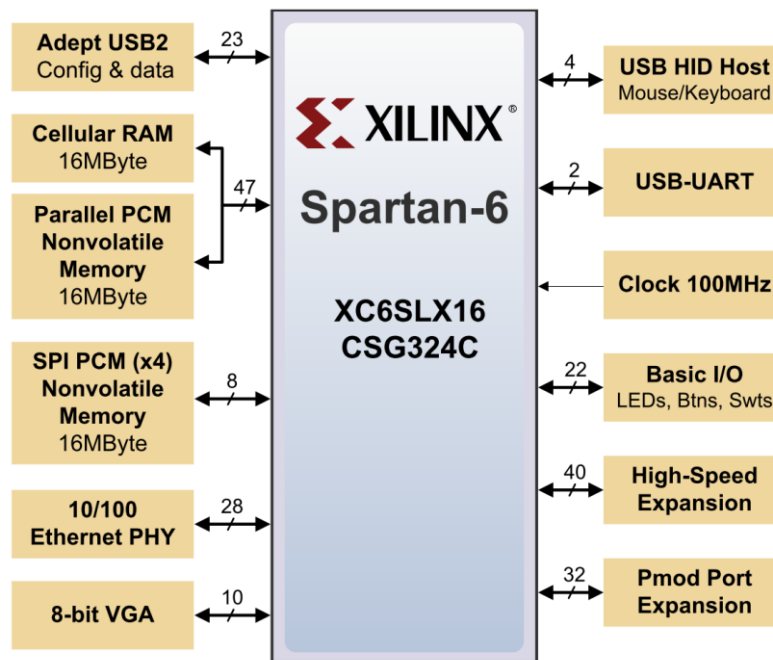


Ilustración 7: Diagrama de componentes de la placa Digilent Nexys3

- **Digilent Atilys:** Esta placa cuenta con la FPGA Spartan 6 LX45, siendo de gama media dentro de la serie Spartan 6. El total de celdas disponibles son 43.661, 401 Kbits de memoria RAM distribuida, 58 unidades DSP48A1, y un total de 2.088 Kbits de memoria RAM de bloque disponibles. Además de estas características, la placa cuenta con 128MB de memoria RAM de tipo DDR2, conector Ethernet, dos puertos de entrada HDMI y dos de salida, códec de audio AC-97, y dos puertos USB como dispositivos UART y HID. Con estas características, resulta fácil pensar que esta placa es capaz de albergar un ordenador personal completo. Como añadido, esta placa permite verificar el consumo eléctrico instantáneo, con lo que podremos obtener buenos análisis de este tipo. En la Ilustración 8 se puede ver el diseño de la placa^[29]:

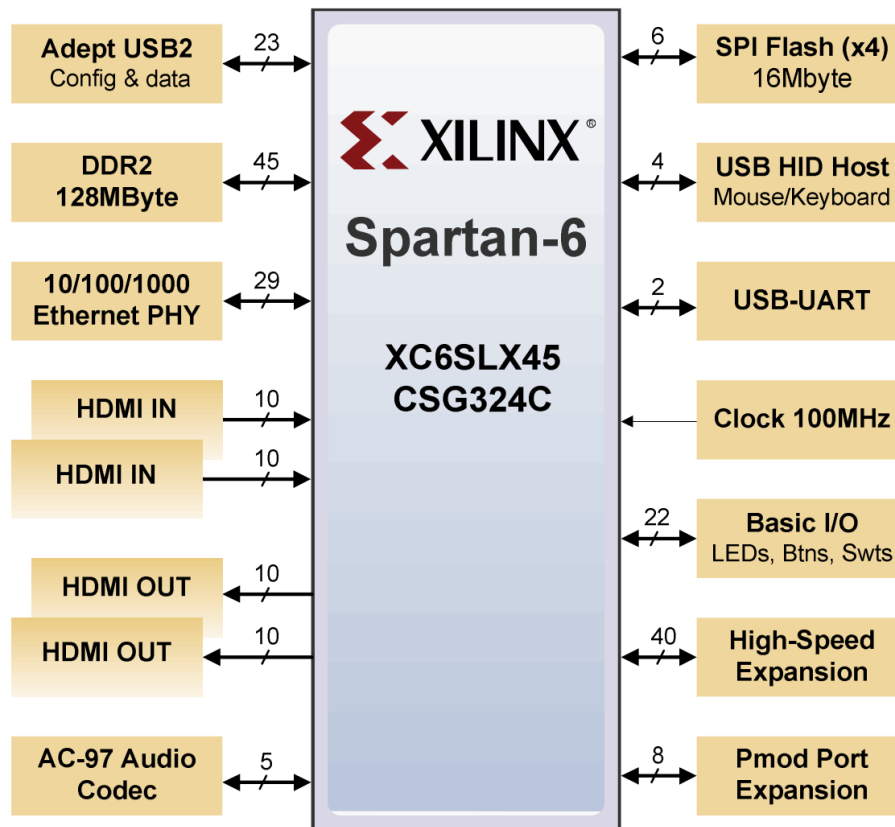


Ilustración 8: Diagrama de componentes de la placa Digilent Atilys

En cuanto al entorno de diseño y desarrollo, utilizaremos los proporcionados por Xilinx. Estos vienen agrupados en la suite Xilinx ISE Design Suite, en su versión 13.3, en la cual está asegurada la compatibilidad con ambas placas. Principalmente, se utilizarán dos IDEs distintos de la suite, debido a que nuestro diseño tendrá componentes hardware y software.

El primero de ellos es el **Xilinx Platform Studio**, o también conocido como XPS. Con él crearemos las distintas plataformas hardware, con los dispositivos e IP cores necesarios para tener la arquitectura MicroBlaze en funcionamiento. Además, en él configuraremos las distintas opciones del soft-processor, con el objetivo de cumplir las especificaciones del proyecto.

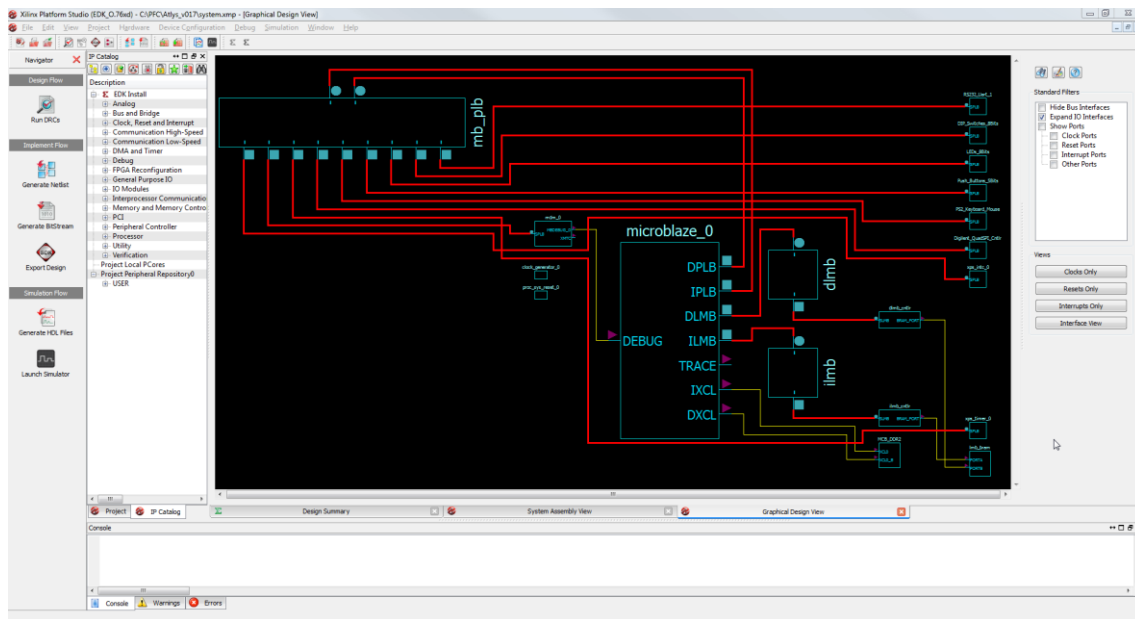


Ilustración 9: Ejemplo de sistema empotrado desarrollado en el Xilinx Platform Studio

El segundo IDE es el **Xilinx Software Development Kit**, con él crearemos los benchmarks, programaremos las especificaciones hardware obtenidas, y obtendremos los resultados pertinentes. Como podemos observar en la siguiente imagen, el entorno de desarrollo está basado en Eclipse, por lo que su uso resultará sencillo y fácil, además de facilitar en gran medida las tareas a realizar.

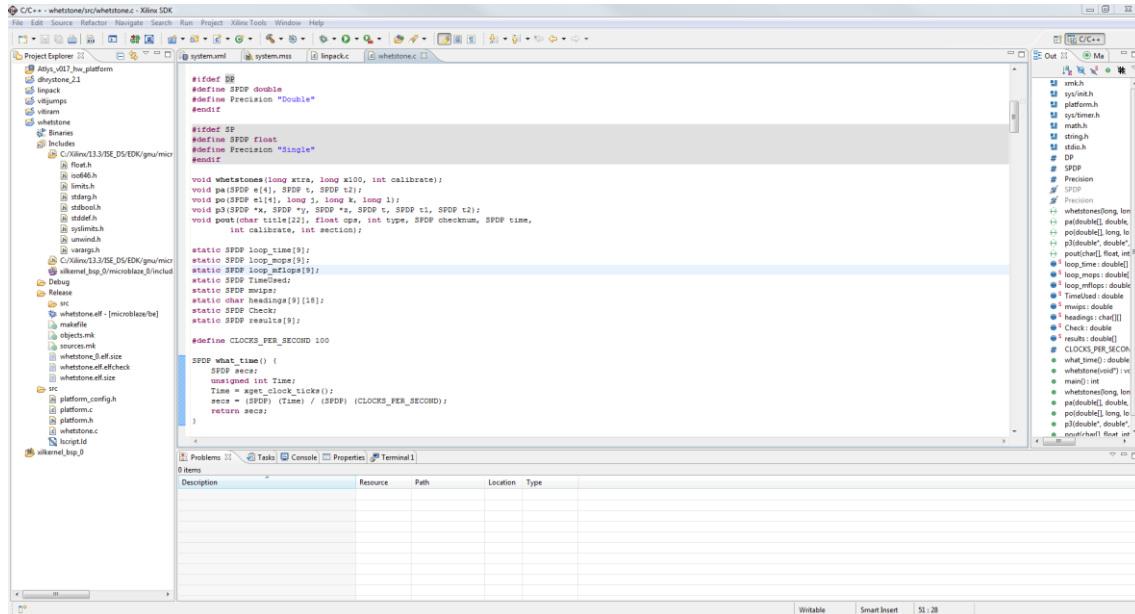


Ilustración 10: Ejemplo de producción de software en el Xilinx Software Development Kit

Con la plataforma de trabajo ya escogida, es posible comenzar el análisis de las necesidades del proyecto en términos de software.

3.2 Descripción General

Como introducción antes de la enumeración de requisitos, se procede a enumerar las capacidades generales del trabajo a realizar, así como sus restricciones y el entorno que influye en la realización del proyecto.

3.2.1 Capacidades generales

Ahora vamos a estudiar cuál o cuales son los principales objetivos que deben cumplir las pruebas de rendimiento de cara a los usuarios. Este es el listado de capacidades que se han de cumplir para el desarrollo del proyecto:

- Las pruebas de rendimiento deben ser capaces de evaluar la **capacidad de cómputo** de la arquitectura MicroBlaze.
- Las pruebas de rendimiento de cómputo serán capaces de evaluar el **rendimiento** de los distintos **elementos que integran el procesador** o soft-processor, incluyendo la ALU, unidades de coma flotante o vectorial y unidades de control y predicción de saltos.

- Las pruebas de rendimiento serán capaces de evaluar el **rendimiento de la memoria** de la arquitectura MicroBlaze.
- Las pruebas de rendimiento serán capaces de realizar un proceso de calibración previo a la ejecución del benchmark, con el objetivo de conseguir **resultados estables**.
- Las pruebas de rendimiento serán capaces de ejecutar distintas **iteraciones** del mismo benchmark, con el objetivo de obtener la media de todos los resultados posibles.
- Las pruebas de rendimiento serán capaces de **mostrar los resultados** obtenidos por pantalla o por consola.

3.2.2 Restricciones generales

En este apartado se van a completar las restricciones que se van a dar para todas las pruebas de rendimiento, y que son necesarias para la correcta evaluación de la arquitectura:

- Los evaluadores de rendimiento se **centrarán** exclusivamente en una parte de la **arquitectura**, evitándose en la medida de lo posible la utilización de benchmarks de evaluación múltiple.
- Se buscará que el **impacto de los relojes** del sistema RTC sea **mínimo**, con el objetivo de que los resultados sean independientes a la precisión del mismo.
- Las pruebas de rendimiento estarán escritas en los lenguajes de programación **C** o **C++**.
- Las pruebas de rendimiento serán **compatibles** con distintas plataformas y arquitecturas, incluyendo los entornos XilKernel, Microsoft DOS, Linux y Microsoft Windows NT, y los procesadores con arquitectura x86, MIPS, ARM y MicroBlaze.
- El **tamaño** y diseño de las pruebas de rendimiento deberá ser **lo mínimo posible**, con el objetivo de evaluar realmente la arquitectura.

3.2.3 Entorno operacional

Las distintas pruebas de rendimiento se ejecutarán en distintas arquitecturas, por lo que éstas tendrán que ser compatibles con los siguientes dispositivos:

Plataforma de pruebas arquitectura MicroBlaze
Placas de desarrollo Digilent Nexys3 y Digilent Atlys, con FPGA integrada Xilinx de la serie Spartan 6 LX y arquitectura MicroBlaze .
Tamaños de memoria RAM de 16Mb y 128Mb .
El sistema operativo que contempla es el XilKernel , con PetaLinux opcional en la placa Digilent Atlys.

Tabla 2: Entorno operacional Digilent

Plataforma de pruebas con arquitectura x86
Ordenadores personales basados en los procesadores Intel 80486DX2 a 66Mhz, Intel 80486DX4 a 75Mhz, Intel Pentium P5 a 60Mhz e Intel Pentium P54C a 75Mhz.
Total de 64 Mb de memoria RAM .
Sistema operativo Microsoft DOS 6.0

Tabla 3: Entorno operacional arquitectura x86

Plataforma de pruebas con arquitectura MIPS
Plataforma portátil de videojuegos Sony Playstation Portable , con procesador MIPS Allegrex con frecuencia regulable de 1 a 333 MHz, basado en el procesador MIPS R4000 de 32 bits.
Dispone de un total de 24 Mb de memoria RAM .
Sistema operativo propietario de Sony. ^[45]

Tabla 4: Entorno operacional arquitectura MIPS

Plataforma de pruebas con arquitectura ARM
Plataforma portátil de videojuegos Nintendo DS , con procesador ARM946E-S y arquitectura ARM9 (juego de instrucciones ARMv5TE) de 32 bits, y una frecuencia de 66 MHz.
Consta de un total de 4 Mb de memoria RAM .
Carece de sistema operativo propio. ^[43]

Tabla 5: Entrono operacional arquitectura ARM

3.3 Requisitos de usuario

El objetivo de este apartado es detallar los requisitos de usuario, con el objetivo de definir, concretar, ordenar y catalogar las distintas necesidades que tenemos para las pruebas de rendimiento.

Estos requisitos se subdividen en requisitos de capacidad, y requisitos de restricción, los primeros indican qué debe ser capaz de realizar las pruebas de rendimiento, y los segundos especifican las restricciones que establecen el cómo se realizan las pruebas de rendimiento.

El formato utilizado en la especificación es el siguiente:

Identificador	XXX-YY
Título	
Prioridad	<input type="checkbox"/> Alta <input type="checkbox"/> Media <input type="checkbox"/> Baja
Necesidad	
Fuente	
Descripción	

Tabla 6: Plantilla de requisitos

Descripción de cada elemento de la plantilla de requisitos:

- **Identificador:** código único e identificativo de cada requisito. Consta de dos partes, XXX se refiere al tipo de requisito siendo RUC los requisitos de usuario, y RUR los requisitos de restricción. La parte YY es el número de requisito.
- **Título:** es el nombre único del requisito.
- **Prioridad:** establece la importancia del requisito, en función del desarrollo de las pruebas de rendimiento. Los valores admitidos son alta, media o baja.

- **Necesidad:** es la importancia del requisito desde el punto de vista del cliente. Los valores posibles son esencial, conveniente u opcional.
- **Fuente:** procedencia del requisito.
- **Descripción:** aclaración completa del significado del requisito.

3.3.1 Requisitos de Capacidad

Identificador	RUC-01
Título	Analizar rendimiento de la unidad de enteros del procesador
Prioridad	<input checked="" type="checkbox"/> Alta <input type="checkbox"/> Media <input type="checkbox"/> Baja
Necesidad	Esencial
Fuente	Cliente
Descripción	
Las pruebas de rendimiento pueden evaluar independientemente la capacidad de cómputo de la unidad de enteros (ALU) del procesador	

Tabla 7: RUC-01

Identificador	RUC-02
Título	Analizar rendimiento de la unidad de coma flotante del procesador
Prioridad	<input checked="" type="checkbox"/> Alta <input type="checkbox"/> Media <input type="checkbox"/> Baja
Necesidad	Esencial
Fuente	Cliente
Descripción	
Las pruebas de rendimiento pueden evaluar independientemente la capacidad de cómputo de la unidad de coma flotante (FPU) del procesador	

Tabla 8: RUC-02

Identificador	RUC-03
Título	Analizar rendimiento de la unidad de predicción de saltos del procesador
Prioridad	<input checked="" type="checkbox"/> Alta <input type="checkbox"/> Media <input type="checkbox"/> Baja
Necesidad	Esencial
Fuente	Cliente
Descripción	
Las pruebas de rendimiento pueden evaluar independientemente la capacidad de ejecución de saltos del procesador	

Tabla 9: RUC-03

Identificador	RUC-04
Título	Analizar rendimiento de transferencias de datos en memoria caché de nivel L1
Prioridad	<input checked="" type="checkbox"/> Alta <input type="checkbox"/> Media <input type="checkbox"/> Baja
Necesidad	Esencial
Fuente	Cliente
Descripción	
Las pruebas de rendimiento pueden evaluar el ancho de banda disponible en la caché de nivel 1 del procesador, tanto en escrituras como en lecturas	

Tabla 10: RUC-04

Identificador	RUC-05
Título	Analizar rendimiento de transferencias de datos en memoria caché de nivel L2
Prioridad	<input checked="" type="checkbox"/> Alta <input type="checkbox"/> Media <input type="checkbox"/> Baja
Necesidad	Esencial
Fuente	Cliente
Descripción	
Las pruebas de rendimiento pueden evaluar el ancho de banda disponible en la caché de nivel 2 del procesador, tanto en escrituras como en lecturas	

Tabla 11: RUC-05

Identificador	RUC-06
Título	Analizar rendimiento de transferencias de datos en memoria RAM
Prioridad	<input checked="" type="checkbox"/> Alta <input type="checkbox"/> Media <input type="checkbox"/> Baja
Necesidad	Esencial
Fuente	Cliente
Descripción	
Las pruebas de rendimiento pueden evaluar el ancho de banda disponible en la memoria RAM del sistema empotrado, tanto en escrituras como en lecturas	

Tabla 12: RUC-06

Identificador	RUC-07
Título	Calibración previa
Prioridad	<input checked="" type="checkbox"/> Alta <input type="checkbox"/> Media <input type="checkbox"/> Baja
Necesidad	Esencial
Fuente	Cliente
Descripción	
Las pruebas de rendimiento ejecutan un proceso de calibrado previo a la ejecución	

Tabla 13: RUC-07

Identificador	RUC-08
Título	Mostrar resultados por pantalla
Prioridad	<input checked="" type="checkbox"/> Alta <input type="checkbox"/> Media <input type="checkbox"/> Baja
Necesidad	Esencial
Fuente	Cliente
Descripción	
Las pruebas de rendimiento muestran los procesos de calibrado y resultados obtenidos desglosados por pantalla, bien sea por comunicaciones por puerto serie o nativamente	

Tabla 14: RUC-08

Identificador	RUC-09
Título	Comprobación de resultados
Prioridad	<input checked="" type="checkbox"/> Alta <input type="checkbox"/> Media <input type="checkbox"/> Baja
Necesidad	Esencial
Fuente	Cliente
Descripción	
Los resultados obtenidos son verificados y contrastados con los resultados que deberían devolver, en el caso de que la prueba de rendimiento deba generar unos resultados determinados	

Tabla 15: RUC-09

3.3.2 Requisitos de Restricción

Identificador	RUR-01
Título	Lenguajes de programación
Prioridad	<input checked="" type="checkbox"/> Alta <input type="checkbox"/> Media <input type="checkbox"/> Baja
Necesidad	Esencial
Fuente	Cliente
Descripción	
Las pruebas de rendimiento únicamente podrán estar escritas en los lenguajes C o C++, utilizando únicamente la librería base que proporcionan.	

Tabla 16: RUR-01

Identificador	RUR-02
Título	Compiladores
Prioridad	<input checked="" type="checkbox"/> Alta <input type="checkbox"/> Media <input type="checkbox"/> Baja
Necesidad	Esencial
Fuente	Cliente
Descripción	
Las pruebas de rendimiento deben ser compilables bajo las versiones superiores a GCC 4.0, OpenWatcom 1.9 y Clang 3.1.	

Tabla 17: RUR-02

Identificador	RUR-03
Título	Espacio en memoria
Prioridad	<input checked="" type="checkbox"/> Alta <input type="checkbox"/> Media <input type="checkbox"/> Baja
Necesidad	Esencial
Fuente	Cliente
Descripción	
Las pruebas de rendimiento deben ocupar el mínimo tamaño posible en memoria RAM, a excepción de las pruebas de rendimiento sobre caché y memoria RAM.	

Tabla 18: RUR-03

Identificador	RUR-04
Título	Uso de código en ensamblador
Prioridad	<input checked="" type="checkbox"/> Alta <input type="checkbox"/> Media <input type="checkbox"/> Baja
Necesidad	Esencial
Fuente	Cliente
Descripción	
Las pruebas de rendimiento deben ser compilables con todas las arquitecturas posibles, por lo que se prohíbe el uso de cualquier tipo de código que incluya primitivas o secciones de código escritas en lenguajes ensamblador.	

Tabla 19: RUR-04

Identificador	RUR-05
Título	Aleatoriedad
Prioridad	<input checked="" type="checkbox"/> Alta <input type="checkbox"/> Media <input type="checkbox"/> Baja
Necesidad	Esencial
Fuente	Cliente
Descripción	
Está prohibido el uso de código basado en iteraciones o saltos aleatorios, a excepción de la inicialización de datos no vinculantes.	

Tabla 20: RUR-05

Identificador	RUR-06
Título	Precisión de los resultados
Prioridad	<input checked="" type="checkbox"/> Alta <input type="checkbox"/> Media <input type="checkbox"/> Baja
Necesidad	Esencial
Fuente	Cliente
Descripción	
El tiempo de ejecución mínimo de las pruebas de rendimiento es de 1 minuto, y de al menos una iteración.	

Tabla 21: RUR-06

Identificador	RUR-07
Título	Arquitecturas soportadas
Prioridad	<input checked="" type="checkbox"/> Alta <input type="checkbox"/> Media <input type="checkbox"/> Baja
Necesidad	Esencial
Fuente	Cliente
Descripción	
Las pruebas de rendimiento deben ser compatibles con la arquitectura de soft-processors MicroBlaze, y adicionalmente las arquitecturas x86, MIPS y ARM.	

Tabla 22: RUR-07

3.4 Medidas de rendimiento

Con los requisitos ya analizados y definidos, vamos a analizar en profundidad las medidas con las que vamos a comparar las distintas arquitecturas. Este aspecto es muy importante, puesto que estas medidas son las que dictaminarán si una arquitectura es eficiente o no, comparándola con el resto de arquitecturas. Cabe destacar que estas medidas deberán ser válidas e iguales para todas las arquitecturas.

Las medidas fundamentalmente han de cumplir las siguientes características conjuntamente a los resultados^[40]:

- **Linealidad:** Los resultados han de ser proporcionales al rendimiento real.
- **Fiabilidad:** Los resultados han de ser fiables, es decir, que éstos sean correctos.
- **Repetibles:** Es importante que los resultados se puedan conseguir más de una vez, y que éstos sean muy parecidos entre sí.
- **Consistentes:** Los resultados han de tener una definición fija y unidades fijas para todos los sistemas.
- **Independientes:** El resultado ha de ser independiente de las marcas que fabriquen un mismo producto, como pueda ser en el caso de los ordenadores personales.
- **Fácil de medir:** Obviamente, una medida difícil de medir dificulta la labor de evaluación de las arquitecturas.

3.4.1 Capacidad de cómputo

La capacidad de cómputo de un procesador está intrínsecamente relacionada con la cantidad de instrucciones que puede emitir por segundo. Concretamente, las medidas que vamos a utilizar son MIPS (millones de instrucciones por segundo) y MFLOPS (millones de operaciones en coma flotante por segundo), y éstas indican la cantidad de operaciones de un determinado tipo que puede realizar un procesador aproximadamente por segundo^[41].

La medida **MIPS** se centra en las operaciones que puede realizar la unidad o unidades ALU del procesador, mientras que la medida **MFLOPS** se centra en las operaciones que pueden realizar la unidad o unidades de coma flotante del procesador (FPU).

Otra posibilidad sería evaluar la frecuencia con la que trabajan las distintas arquitecturas, pero esta medida está demostrada de no representar fielmente el rendimiento de una arquitectura. Se dan casos en los que procesadores a menor frecuencia rinden mejor que un procesador a mayor frecuencia, debido principalmente al uso de un conjunto de instrucciones más avanzado, o por tener una implementación mejor de los mismos.

Por último, se ha de destacar que la medida de número de ciclos por instrucción no se va a utilizar por la dificultad que implica evaluar esta medida, al necesitar de hardware especializado para realizar este cometido. Sin embargo, en los análisis de las pruebas se utilizarán los datos existentes de rendimiento en términos CPI en diversa documentación para contrastar los resultados.

3.4.2 Acceso a memoria

Al igual que la capacidad de cómputo, el acceso a memoria puede medirse en la frecuencia en la que trabaja la memoria, sin embargo esta tampoco es un indicativo real del rendimiento de la misma. Para evitar este problema, evaluaremos el rendimiento en la **cantidad de datos que se puedan transferir por unidad de tiempo**. Un ejemplo de esta medida serían MB/s o lo que es lo mismo, megabytes por segundo.

Otra posibilidad de medida de rendimiento en el acceso a memoria sería la evaluación de las **latencias de acceso a memoria**. Existen cuatro puntos principales para evaluar las latencias en memoria de tipo DRAM:

- **Latencia CAS:** Es el tiempo necesitado para que el controlador de memoria pueda enviar la dirección de la columna y recibir el primer bit de la fila abierta por el controlador.

- **Retardo entre dirección de fila y dirección de columna:** Es el número de ciclos de reloj necesarios entre la abertura de una hilera de columnas de memoria y el acceso a los datos.
- **Tiempo de precarga de fila:** Es el número de ciclos de reloj necesitados entre la emisión de la orden de precarga y la apertura de la siguiente fila.
- **Tiempo de activación de fila:** Tiempo en ciclos de reloj necesarios entre la orden de activación del banco de memoria y la orden de precarga.

Sin embargo para realizar estas mediciones también es necesario disponer de hardware complejo, además de representar parcialmente el rendimiento que se puede alcanzar. Por ello, simplemente evaluaremos la capacidad de transferir datos por unidad de tiempo, dado que es una medida fiel al rendimiento de la arquitectura.

3.4.3 Consumo de energía

El último aspecto que se va a evaluar en las plataformas es el consumo energético. Hoy en día este es uno de los aspectos más importantes en los sistemas empujados, como puedan ser los dispositivos móviles o tabletas. Para medir el consumo de energía (potencia), se emplea la medida estándar, que es el **número de vatios** que consume un dispositivo.

Esta medida se aplicará también al rendimiento, evaluándose el **rendimiento por megahertzio** y el **rendimiento por vatio**, lo que nos dará un indicativo de lo eficiente que son las distintas plataformas.



4 Diseño

En este capítulo se va a mostrar el diseño arquitectónico de las distintas pruebas de rendimiento que se van a crear para este proyecto. Además, se mostrará el diseño detallado de cada una de ellas, y una explicación de las decisiones tomadas a la hora de realizar el diseño.

4.1 Arquitectura de los benchmarks

En este apartado se muestra el diseño de la arquitectura básica de las distintas pruebas de rendimiento. Dado que las pruebas de rendimiento pueden evaluar distintas partes de la arquitectura, se ha generado una arquitectura reutilizable, la cual con modificaciones sencillas permite crear nuevos benchmarks, enfocados en otros aspectos o evaluando cosas distintas. La arquitectura que se va a seguir en todas los benchmarks para este proyecto es la siguiente:

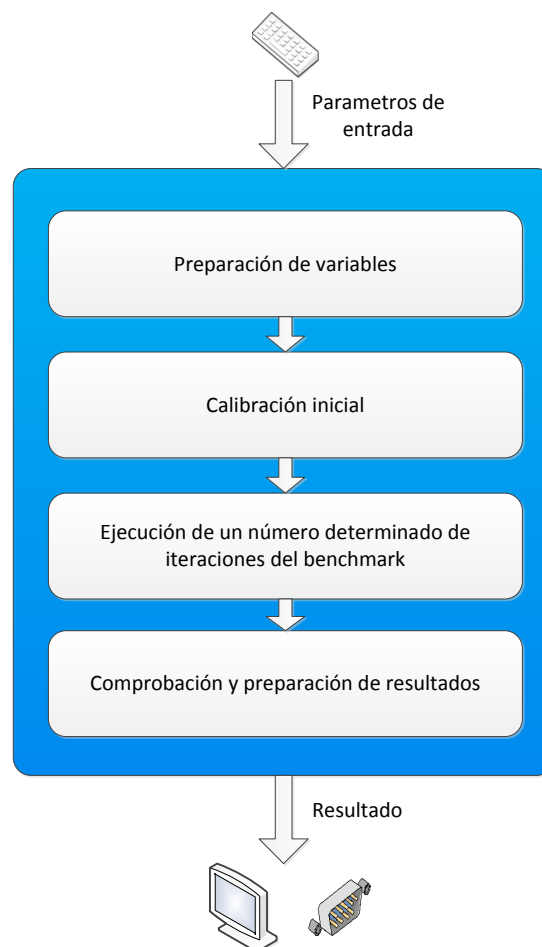


Ilustración 11: Arquitectura estándar para los benchmarks

Tal y como se puede apreciar en la figura anterior, la arquitectura del sistema está pensada para la ejecución de las pruebas de rendimiento en una sola máquina o sistema empotrado, sin realizar comunicaciones con otros dispositivos. Esto significa que únicamente vamos a disponer de un componente por cada prueba de rendimiento, de fácil reutilización, y con distintos subcomponentes que preparan el conjunto de las pruebas. De esta manera se evita disponer de código complejo, que pueda alterar o influir negativamente en las pruebas.

Además, se ha evitado el uso de patrones arquitectónicos, como puedan ser el Modelo Vista Controlador (MVC), debido a que no nos interesa centrarnos en la funcionalidad o la posibilidad de ampliación de las pruebas, sino en la **optimización y ligereza** del código. Por otro lado, la restricción de utilizar código en C o C++ dificulta en gran medida esta posibilidad.

4.2 Diseño detallado

En el diseño detallado se va a mostrar el diseño y funcionamiento interno de los distintos benchmarks que vamos a implementar, así como el diseño de los resultados que proporcionan. Es destacable que todos ellos han sido correspondientemente ajustados a la arquitectura propuesta en el apartado anterior. Primeramente vamos a listar las distintas pruebas de rendimiento a implementar, y los motivos por los que han sido elegidos:

- **Dhrystone 2.1:** Evalúa la capacidad de **procesamiento** de la unidad **ALU** del microprocesador, y está ampliamente reconocido. Mejora diversas deficiencias existentes respecto de la versión 1.0.
- **Linpack:** Evalúa la capacidad de **procesamiento** de la unidad de **coma flotante** del microprocesador a través de una serie de librerías matemáticas. También es ampliamente reconocido.
- **Whetstone:** Al igual que Linpack, evalúa la capacidad de **procesamiento** de la unidad de **coma flotante**, además de funcionalidades diversas de la **ALU**. Al igual que los dos anteriores, es muy usado como medida de rendimiento en procesadores.
- **Benchmark de diseño propio para predicción de saltos:** No se ha encontrado ningún benchmark de tamaño mínimo que esté exclusivamente dedicado a la evaluación de las **predicciones de saltos**, por lo que se ha decidido crear uno nuevo, en el que se evalúe específicamente desde distintos puntos de vista esta funcionalidad. A este benchmark se le ha llamado **VitiJumps**.

- **Benchmark de diseño propio para evaluación de rendimiento de memoria RAM:** Actualmente existen gran cantidad de este tipo de benchmarks, pero con la problemática de estar orientados para sistemas operativos complejos con interfaz gráfica (como Microsoft Windows), además cuentan con gran cantidad de código en ensamblador exclusivo para determinadas arquitecturas (sobre todo x86 y ARM). Es por ello que se va a crear uno propio, sin utilizar código en ensamblador y que sea capaz de ser reutilizable. A este benchmark se le ha llamado **VitiRAM**.

Los tres primeros ya existen y están implementados, pero los vamos a analizar en profundidad, y adaptar para su uso en este proyecto. Cabe destacar que estos tres benchmarks son fáciles de migrar a otras plataformas, por lo que en este aspecto resultan muy interesantes. Concretamente, se han utilizado las versiones diseñadas y adaptadas por Roy Longbottom^[42], uno de los principales creadores del benchmark Whetstone, y consultor de varias empresas de informática.

4.2.1 Dhrystone 2.1

El diseño de este benchmark esta basado en una estructura simple de tres ficheros, dos siendo código fuente en C y una cabecera para ambas fuentes. Los ficheros son los siguientes:

- **Dhry.h:** Tiene las definiciones de datos globales, y los comentarios.
- **Dhry_1.c:** Contiene el código correspondiente a la versión original programada en ADA, denominada Pack_1. Incluye el código principal del programa (método *main*), y los procedimientos a evaluar denominadas Proc_X, siendo X el valor 1 hasta 5 inclusive.
- **Dhry_2.c:** Contiene el código correspondiente a la versión original programada en ADA, denominada Pack_2. Incluye los procedimientos a evaluar Proc_X, siendo X el valor 6 hasta 8 inclusive, y las funciones Func_Y, siendo Y el valor 1 hasta 3 inclusive.

Dado que el diseño del benchmark Dhrystone 2.1 es bastante complejo internamente, sólo se va a incluir una estadística del tipo de operaciones que se incluyen en dicho benchmark. En su totalidad, se puede determinar que un 58% de las instrucciones ejecutadas son asignaciones, un 28% control de declaraciones y los procedimientos y llamadas a funciones un 15%. Cabe destacar que todas las instrucciones están dinámicamente balanceadas.

Las operaciones totales son 101, divididas de la manera que indica la Tabla 23:

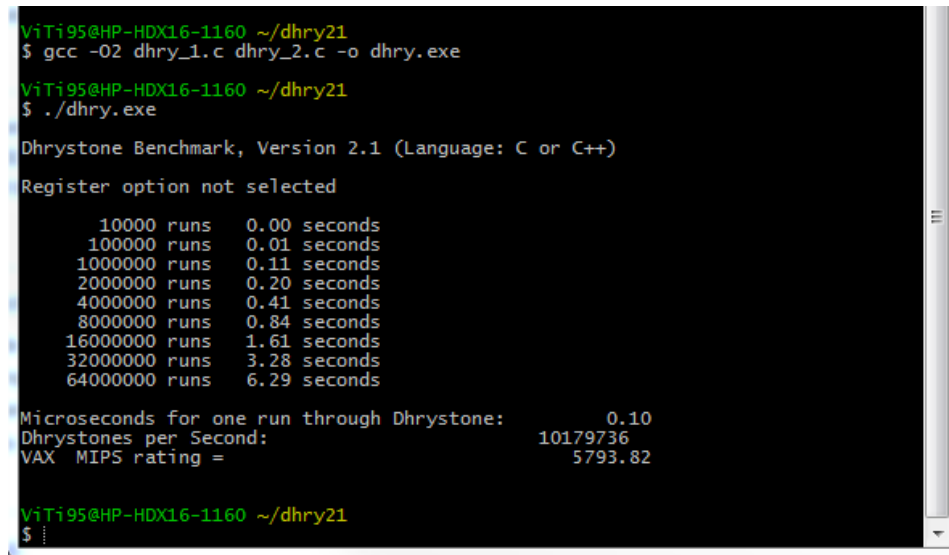
Tipo de declaración	Numero de veces
V1 := V2	15
V := Constante	12
Asignación con elemento de array	7
Asignación con estructura record	6
X := Y + - and or Z	5
X := Y + - = constante	6
X := + - 1	3
X := Y * / Z	2
X := expresión con dos operadores	1
X := expresión con tres operadores	1
If then	7
If then else	7 (3 ejecutadas, 4 no ejecutadas)
For I in 1..N do	7
While do	4
Repeat until	1
Case end	1
With	1
Llamada a procedimiento	10
Llamada a función con asignación	5

Tabla 23: Operaciones y tipos en Dhrystone 2.1

Por otro lado, cabe destacar que este número de operaciones ejecutadas, puede ser modificado por el compilador si éste estima que existen operaciones redundantes/inexistentes, por lo que los resultados que proporciona este benchmark pueden depender en gran medida del compilador^{[36][37][38]}.

El resultado que ofrece este benchmark es estimativo, y en él se indican el número de operaciones Dhrystone (todo el conjunto de instrucciones que ejecuta el benchmark) que el sistema realiza por segundo (conocidos como DMIPS, medido en millones de operaciones por segundo), y otro resultado en el que compara dicho resultado con el obtenido por la maquina VAX 11/780, la cual era capaz de obtener el resultado de 1 DMIPS.

Existe además la opción de mostrar el resultado de las distintas funciones que ejecuta el benchmark, y compararlo con el resultado que se debería obtener. La Ilustración 12 es un ejemplo del resultado obtenido con Dhrystone 2.1 compilado para la plataforma Microsoft Windows a través de Cygwin:



```

ViTi95@HP-HDX16-1160 ~/dhry21
$ gcc -O2 dhry_1.c dhry_2.c -o dhry.exe

ViTi95@HP-HDX16-1160 ~/dhry21
$ ./dhry.exe

Dhrystone Benchmark, Version 2.1 (Language: C or C++)
Register option not selected

      10000 runs      0.00 seconds
     100000 runs      0.01 seconds
    1000000 runs      0.11 seconds
    2000000 runs      0.20 seconds
    4000000 runs      0.41 seconds
    8000000 runs      0.84 seconds
   16000000 runs      1.61 seconds
   32000000 runs      3.28 seconds
   64000000 runs      6.29 seconds

Microseconds for one run through Dhrystone:      0.10
Dhrystones per Second:      10179736
VAX MIPS rating =      5793.82

ViTi95@HP-HDX16-1160 ~/dhry21
$

```

Ilustración 12: Ejemplo de ejecución de Dhrystone 2.1

Del ejemplo el resultado final es el indicado en “VAX MIPS rating”, y los pasos anteriores corresponden a la calibración propia del benchmark.

4.2.2 Linpack

El benchmark Linpack basa su diseño interno en un conjunto de funciones matemáticas orientadas a la evaluación del rendimiento computacional en coma flotante. El cometido principal es evaluar cómo de rápido es capaz de solucionar un sistema de ecuaciones lineales del tipo $Ax = b$, de tamaño $N \times N$. Concretamente, la versión utilizada para este proyecto es la denominada Linpack 100, que utiliza un sistema 100×100 ^{[33][34]}.

La resolución del sistema de ecuaciones requiere un total de $O(n^3)$ operaciones en coma flotante, concretamente $\frac{2}{3}n^3 + 2n^2 + O(n)$ operaciones de suma y multiplicaciones. Es por ello que el tiempo requerido para la resolución de dicho problema es de aproximadamente:

$$time_n = \frac{time_{100} \cdot n^3}{100^3}$$

El funcionamiento interno se basa en dos rutinas, DGEFA y DGESL, la primera realiza la descomposición mediante pivotes, y la segunda utiliza dicha descomposición para solucionar el sistema de ecuaciones lineales dado. En la función DGEFA se invierte la mayor parte del tiempo del benchmark, al tener una complejidad $O(n^3)$, mientras que en DGESL se tiene una complejidad de $O(n^2)$.

Además de estas dos rutinas, existen otras tres pertenecientes a la librería BLAS, denominadas DAXPY, IDAMAX y DSCAL, y que son utilizadas por las librerías DGEFA y DGESL:

- **DAXPY:** Realiza la operación de multiplicar el escalar α al vector X, y añadir dicho resultado al vector Y. Se utiliza en el 90% del total de tiempo de ejecución, y es llamado $n^2/2$ veces por DGEFA, y $2n$ veces por DGESL.
- **IDAMAX:** Calcula el índice del elemento de un vector que tenga el modulo mayor, en vectores de longitud variable.
- **DSCAL:** Realiza un total de N multiplicaciones en coma flotante.

El número total de operaciones realizadas por la función DGEFA en Linpack 100 es la siguiente:

Tipo de operación	Número de veces
Suma	328350
Multiplicación	333300
Reciproca	99
Valor absoluto	5364
Menor o igual que	4950
Distinto de cero	5247

Tabla 24: Operaciones y tipos en DGEFA de Linpack 100x100

Por otro lado, el número total de operaciones realizadas por la función DGESL en Linpack 100 es la siguiente:

Tipo de operación	Número de veces
Suma	9900
Multiplicación	9900
División	100
Negación	100
Distinto de cero	199

Tabla 25: Operaciones y tipos en DGESL de Linpack 100x100

Contando el total de operaciones de las funciones DGEFA y DGESL, obtenemos que en Linpack 100 se realizan un total de 697.509 operaciones en coma flotante.

El diseño de este benchmark ha sido modificado para poder alternar entre precisión simple o doble, y poder utilizar versiones con bucles de código desenrollado manualmente, con el objetivo de conseguir una mejora en el rendimiento y así poder realizar mejores comparaciones (y a su vez poder ver si las optimizaciones del compilador pueden obtener un mejor o peor resultado). Todas las rutinas y funciones han sido incluidas en un único fichero C, con el objetivo de simplificar la estructura del código.

El resultado que ofrece Linpack 100 es indicativo, siendo éste una estimación de la cantidad de operaciones que puede realizar el sistema, medido en MFlops, o lo que es lo mismo, millones de operaciones de coma flotante por segundo. En la Ilustración 13 se muestra un ejemplo del resultado que devuelve:

```

ViTi95@HP-HDX16-1160 ~/linpack
$ ./linpack.exe
Unrolled Double Precision Linpack Benchmark - PC Version in 'C/C++'

norm resid      resid      machep      x[0]-1      x[n-1]-1
  1.9    8.39915160e-14    2.22044605e-16    -6.22835117e-14    -4.16333634e-14

Times are reported for matrices of order      100
1 pass times for array with leading dimension of 201

      dgefa      dgesl      total      Mflops      unit      ratio
    0.00000    0.00000    0.00000         0.00    0.0000    0.0000

Calculating matgen overhead
  10 times    0.00 seconds
 100 times    0.01 seconds
1000 times    0.12 seconds
2000 times    0.22 seconds
4000 times    0.47 seconds
8000 times    0.91 seconds
16000 times    1.81 seconds
32000 times    3.60 seconds
64000 times    7.25 seconds
Overhead for 1 matgen    0.00011 seconds

Calculating matgen/dgefa passes for 5 seconds
  10 times    0.02 seconds
 100 times    0.05 seconds
1000 times    0.53 seconds
2000 times    1.03 seconds
4000 times    2.09 seconds
8000 times    4.18 seconds
16000 times    8.39 seconds
Passes used    9531

Times for array with leading dimension of 201

      dgefa      dgesl      total      Mflops      unit      ratio
    0.00041    0.00016    0.00057    1209.79    0.0017    0.0101
    0.00041    0.00016    0.00057    1206.44    0.0017    0.0102
    0.00042    0.00015    0.00057    1206.44    0.0017    0.0102
    0.00041    0.00015    0.00056    1220.39    0.0016    0.0100
    0.00041    0.00016    0.00056    1216.99    0.0016    0.0101
Average                                1212.01

Calculating matgen2 overhead
Overhead for 1 matgen    0.00011 seconds

Times for array with leading dimension of 200

      dgefa      dgesl      total      Mflops      unit      ratio
    0.00040    0.00016    0.00056    1223.53    0.0016    0.0100
    0.00041    0.00016    0.00057    1209.29    0.0017    0.0101
    0.00041    0.00016    0.00056    1223.76    0.0016    0.0100
    0.00041    0.00016    0.00056    1223.30    0.0016    0.0100
    0.00041    0.00016    0.00056    1223.53    0.0016    0.0100
Average                                1220.68

Unrolled Double Precision    1212.01 Mflops

ViTi95@HP-HDX16-1160 ~/linpack
$

```

Ilustración 13: Ejemplo de ejecución de Linpack 100x100

La salida muestra los distintos procesos de calibración del benchmark, y finalmente el resultado obtenido en MFLOPS.

4.2.3 Whetstone

El funcionamiento y diseño interno de la prueba de rendimiento Whetstone, es muy similar al diseño realizado para Dhrystone 2.1, éste consiste en la ejecución de una determinada serie de funciones, que contienen determinadas instrucciones en coma flotante y otras pertenecientes a la unidad ALU del procesador^[35].

La idea principal es realizar todos los cálculos e iteraciones con valores diferentes en cada ejecución, y ser capaces de hacerlo con una repetición infinita. Para ello, se dispone de varias operaciones matemáticas, agrupadas en módulos. El **primer módulo** se basa en las siguientes funciones matemáticas, y evalúa el rendimiento con elementos en un vector:

$$x1 = (x1 + x2 + x3 - x4) \cdot 0.5$$

$$x2 = (x1 + x2 - x3 + x4) \cdot 0.5$$

$$x3 = (x1 - x2 + x3 + x4) \cdot 0.5$$

$$x4 = (-x1 + x2 + x3 + x4) \cdot 0.5$$

Esta serie de transformaciones permiten converger una secuencia de números x_i en un resultado final que cumple la siguiente característica:

$$x1 = x2 = x3 = x4 = 1.0$$

Debido a que las operaciones en coma flotante simple y doble tienen errores de precisión, si se cambia el valor de la variable 0.5 a 0.499975 y se cambian los valores de $x1$ a 1.0, y $x2, x3, x4$ a -1.0, se consigue obtener una secuencia prácticamente ilimitada de iteraciones con distintos valores.

El **segundo módulo** aplica la misma idea del primero, pero realizando dicha funcionalidad llamando a la función *pa* que contiene el algoritmo matemático, y pasando el array y un valor temporal por parámetro.

La evaluación de elementos condicionales en coma flotante se realiza en el **módulo tercero**. Éste comprueba en un número determinado de iteraciones el valor de j , y dependiendo del resultado en la evaluación condicional, se modifica el valor de j

siguiendo los principios de los dos primeros módulos. Las condiciones a evaluar son de igualdad, menor que y mayor que respecto de una variable constante.

El **cuarto módulo** realiza operaciones simples aritméticas, fácilmente operables por la ALU del procesador. El conjunto de operaciones realizadas para este módulo son sumas, restas y multiplicaciones.

El **quinto módulo** ejecuta funciones trigonométricas complejas, como pueden ser los arcotangentes, senos y cosenos, además de incluir operaciones del tipo, suma, resta y multiplicación. Las funciones ejecutadas en este módulo son las siguientes:

$$x = t \cdot \arctan(2.0 \cdot \sin(x) \cdot \cos(x) / (\cos(x \cdot y) + \cos(x - y) - 1.0))$$

$$y = t \cdot \arctan(2.0 \cdot \sin(y) \cdot \cos(y) / (\cos(x + y) + \cos(x - y) - 1.0))$$

Estas dos funciones están diseñadas para que, utilizando un valor de t igual a 0.499975, y haciendo que x e y tengan valores de 0.5, estas se transformen ligeramente en sí mismas, con lo que se obtiene una variación lenta de las mismas.

Los **módulos 6 y 7** son similares al 2, el primero ejecuta llamadas a la función $p3$ con valores pasados por parámetro y por puntero, que computa varias operaciones aritméticas simples, y el segundo ejecuta llamadas a la función po , que realiza asignaciones de valores en memoria a través de posiciones en arrays.

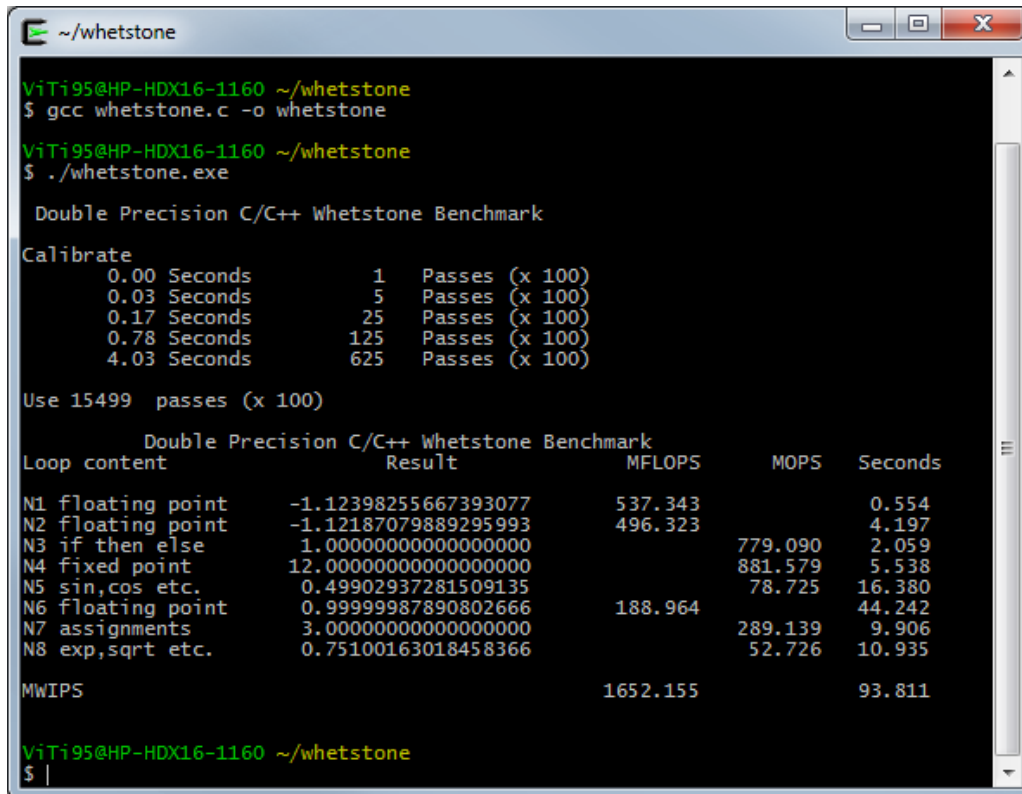
Finalmente el **último módulo** (el número 8), realiza una serie de funciones algebraicas de uso común, como son las raíces cuadradas, las exponenciales y los logaritmos. La fórmula concreta utilizada en este módulo es la siguiente:

$$x = \text{sqrt}\left(\exp\left(\frac{\ln(x)}{t1}\right)\right)$$

Los valores iniciales estiman que $t1$ ha de valer 0.50025, con un valor inicial de x de 0.75, lo que origina una serie de valores estables de x con la ejecución de las distintas iteraciones.

Además, se ha modificado el benchmark para que éste pueda aceptar tanto coma flotante de precisión simple, como coma flotante de precisión doble. En cuanto a los resultados ofrecidos por la prueba, éstos se desglosan por cada módulo, ofreciendo al final un compendio de todos ellos, resumiendo el potencial del ordenador en términos de MWIPS, o lo que es lo mismo, millones de instrucciones Whetstone por segundo.

La Ilustración 14 es un ejemplo de la ejecución del benchmark:



```
~/whetstone
ViTi95@HP-HDX16-1160 ~/whetstone
$ gcc whetstone.c -o whetstone
ViTi95@HP-HDX16-1160 ~/whetstone
$ ./whetstone.exe

Double Precision C/C++ Whetstone Benchmark

Calibrate
0.00 Seconds      1    Passes (x 100)
0.03 Seconds      5    Passes (x 100)
0.17 Seconds     25    Passes (x 100)
0.78 Seconds    125    Passes (x 100)
4.03 Seconds    625    Passes (x 100)

Use 15499 passes (x 100)

Double Precision C/C++ Whetstone Benchmark
Loop content      Result      MFLOPS      MOPS      Seconds
N1 floating point  -1.12398255667393077  537.343      0.554
N2 floating point  -1.12187079889295993  496.323      4.197
N3 if then else    1.00000000000000000  779.090      2.059
N4 fixed point     12.00000000000000000  881.579      5.538
N5 sin,cos etc.    0.49902937281509135  78.725     16.380
N6 floating point  0.99999987890802666  188.964     44.242
N7 assignments    3.00000000000000000  289.139      9.906
N8 exp,sqrt etc.   0.75100163018458366  52.726     10.935

MWIPS
1652.155
93.811

ViTi95@HP-HDX16-1160 ~/whetstone
$ |
```

Ilustración 14: Ejemplo de ejecución de Whetstone

Como se puede observar, en el ejemplo primero se realiza el proceso de calibración, y posteriormente se muestra el resultado de cada módulo de forma desglosada, incluyendo el tiempo que han tardado en ejecutarse. Finalmente, se muestra el resultado final de la prueba en MWIPS.

4.2.4 VitiJumps

Esta prueba de rendimiento ha sido diseñada específicamente para este proyecto. Su cometido es conseguir evaluar la capacidad de un microprocesador en términos de predicción y ejecución de saltos. Los saltos se producen en los conjuntos de instrucciones de tipo *branch* o de tipo *jump* en las distintas arquitecturas, y éstos son reflejados mayoritariamente en lenguajes de alto nivel por las sentencias *if-then-else* y *switch-select-case*.

Este aspecto es muy importante en los procesadores y arquitecturas actuales, puesto que un salto realizado con una predicción equivocada, ocasiona una pérdida de rendimiento elevada, de entorno a unos 10 a 20 ciclos de reloj. Esto ocurre debido a que el flujo de ejecución del programa no es el esperado, y el procesador por tanto debe recargar todas las instrucciones que él tenía determinado que iba a ejecutar (sobre todo aquellos basados en la tecnología de procesadores segmentados, o *pipelines*)^[39].

La problemática de los saltos mal predichos se soluciona con la implantación de predictores de saltos en los procesadores. Es por ello que esta prueba de rendimiento se va a focalizar en la evaluación de saltos.

Para ello, el diseño interno del benchmark está basado en dos bucles de código que evalúan la capacidad de salto de un procesador. La primera de ellas evalúa fácilmente la predicción de saltos simples:

```
for (int i = 0; i < max; i++){  
    if (<condition>)  
        sum++  
    else  
        sum--;  
}
```

Cambiando la condición por una instrucción realmente simple, se puede conseguir que el procesador dedique la mayor parte de tiempo en la ejecución de la rutina de salto (aunque exista un salto adicional en la sección del bucle *for*). La tabla siguiente muestra las condiciones que se evalúan, las cuales condicionarán una serie determinada de fallos o aciertos:

Condición	Patrón
(i & 0x80000000) == 0	Verdadero siempre
(i & 0xFFFFFFFF) == 0	Falso siempre
(i & 1) == 0	Alternancia verdadero y falso
(i & 2) == 0	Dos verdaderos, dos falsos, ...
(i & 4) == 0	Cuatro verdaderos, cuatro falsos, ...
(i & 8) == 0	Ocho verdaderos, ocho falsos, ...
(i & 16) == 0	Dieciséis verdaderos, dieciséis falsos, ...

Tabla 26: Condiciones ejecutadas en el primer módulo de VitiJumps

Con las condiciones descritas en esta tabla podemos ver cómo actuarán los predictores de saltos, pudiendo determinar cuáles son las situaciones en las que se perdería o ganaría rendimiento (a mayor número de iteraciones ejecutadas por unidad de tiempo, mayor rendimiento).

El segundo módulo del programa se dedica a evaluar los saltos indirectos. Para realizar este cometido se ha utilizado una versión modificada del primer módulo. El bucle de código que realiza esta acción es el siguiente:

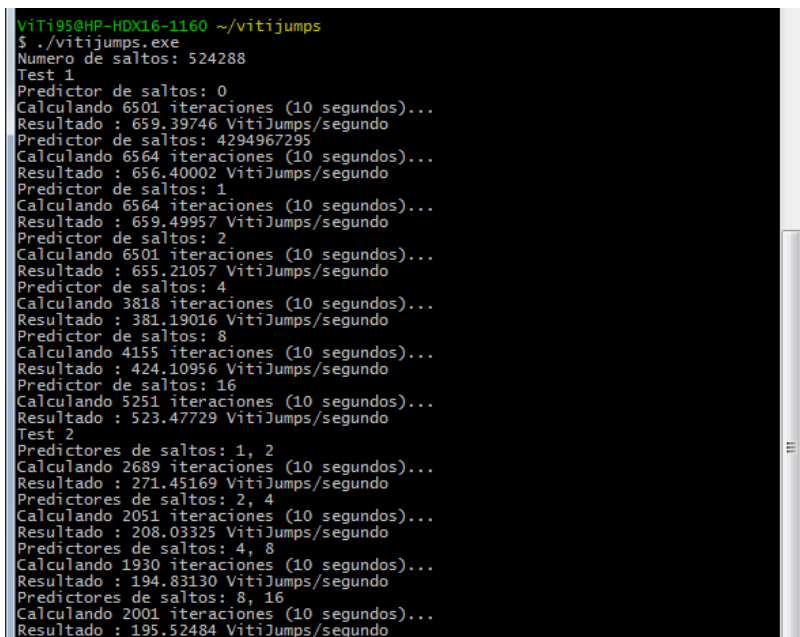
```
for (int i = 0; i < max; i++){
    if (<condition_1>)
        a = 1;
    else
        a = 0;

    if (<condition_2>)
        b = 1;
    else
        b = 0;

    if ((a & b) == 1)
        sum++;
    else
        sum--;
}
```

Utilizando las condiciones descritas en la tabla anterior, e intercalándolas entre las condiciones 1 y 2 de las dos primeras sentencias *if-then-else*, se dificulta en gran medida la predicción del tercer salto.

La evaluación del rendimiento se realiza verificando cuántas veces por segundo es capaz de ejecutar cada bucle de código de los módulos 1 y 2 para las distintas configuraciones de condiciones (la unidad de medida ha sido definida para este benchmark, y es denominada como VitiJumps por segundo). En total se realizan 7 valoraciones para el módulo 1, y 4 para el módulo 2. La Ilustración 15 es un ejemplo de ejecución de la prueba de rendimiento:



```
ViTi95@HP-HDX16-1160 ~/vitiJumps
$ ./vitiJumps.exe
Numero de saltos: 524288
Test 1
Predictor de saltos: 0
Calculando 6501 iteraciones (10 segundos)...
Resultado : 659.39746 VitiJumps/segundo
Predictor de saltos: 4294967295
Calculando 6564 iteraciones (10 segundos)...
Resultado : 656.40002 VitiJumps/segundo
Predictor de saltos: 1
Calculando 6564 iteraciones (10 segundos)...
Resultado : 659.49957 VitiJumps/segundo
Predictor de saltos: 2
Calculando 6501 iteraciones (10 segundos)...
Resultado : 655.21057 VitiJumps/segundo
Predictor de saltos: 4
Calculando 3818 iteraciones (10 segundos)...
Resultado : 381.19016 VitiJumps/segundo
Predictor de saltos: 8
Calculando 4155 iteraciones (10 segundos)...
Resultado : 424.10956 VitiJumps/segundo
Predictor de saltos: 16
Calculando 5251 iteraciones (10 segundos)...
Resultado : 523.47729 VitiJumps/segundo
Test 2
Predictores de saltos: 1, 2
Calculando 2689 iteraciones (10 segundos)...
Resultado : 271.45169 VitiJumps/segundo
Predictores de saltos: 2, 4
Calculando 2051 iteraciones (10 segundos)...
Resultado : 208.03325 VitiJumps/segundo
Predictores de saltos: 4, 8
Calculando 1930 iteraciones (10 segundos)...
Resultado : 194.83130 VitiJumps/segundo
Predictores de saltos: 8, 16
Calculando 2001 iteraciones (10 segundos)...
Resultado : 195.52484 VitiJumps/segundo
```

Ilustración 15: Ejemplo de ejecución de VitiJumps

En el ejemplo se puede observar cuales son los test ejecutados (1 para el de saltos directos, 2 para el de saltos indirectos) y los predictores de saltos evaluados, con su correspondiente resultado.

4.2.5 VitiRAM

La evaluación del rendimiento de la memoria RAM de los distintos sistemas siempre ha estado ligada a la arquitectura en la que se ejecuta. Esto resulta un problema, puesto que para evaluar dicho rendimiento es necesario la inclusión de instrucciones en ensamblador directamente en el código, lo que dificulta en gran medida la implementación de un mismo código en distintas arquitecturas.

Para solucionar este problema, utilizaremos las funciones básicas que incluye C++ para el tratamiento de bloques de memoria, concretamente *memset* y *memcpy*:

- **Memset:** Esta función nos permite rellenar un bloque de memoria con un determinado valor, permitiendo elegir la cantidad de *bytes* que queremos rellenar. Con esta función podremos probar la velocidad de escritura en memoria RAM (y consecuentemente en las cachés de nivel 1, 2 y 3 en el caso de que existan).
- **Memcpy:** Copia un bloque de memoria de una posición a otra, con la posibilidad de elegir la cantidad de *bytes* que queremos copiar. Esta función nos va a permitir evaluar la capacidad de lectura + escritura en memoria RAM.

Teniendo en cuenta estas dos rutinas, y que no existen funciones específicas dedicadas a la lectura única de datos desde memoria RAM, sólo podremos realizar la evaluación de escritura y copia de datos.

La evaluación de rendimiento para escritura de datos se realiza escribiendo el mismo valor sobre la misma posición de memoria, y con un tamaño de bloque de datos determinado durante un determinado tiempo:

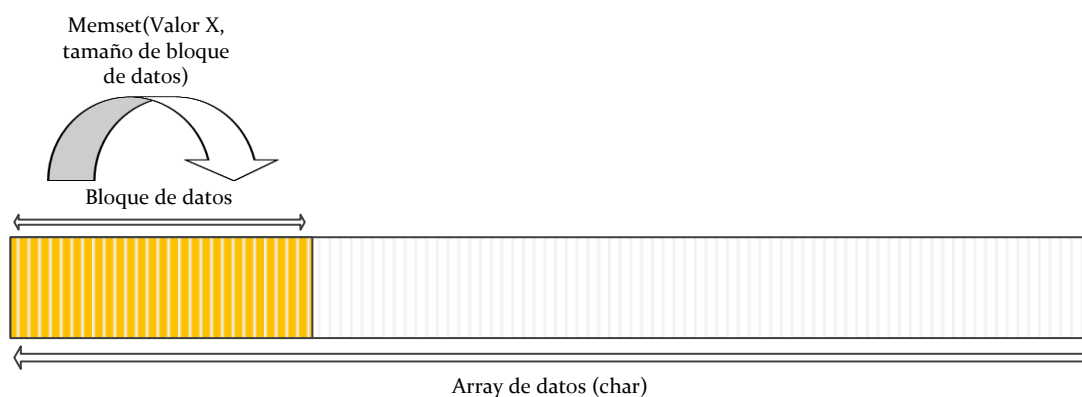


Ilustración 16: Funcionamiento de la evaluación de rendimiento de escritura en memoria

Ejecutando este código, se puede obtener la cantidad de datos que se han escrito por segundo, realizando una simple división del total de datos escritos entre el tiempo que se ha tardado en realizar. La medida por tanto será estándar, y la describiremos como MB/s. Para la evaluación de rendimiento en copia de datos usaremos la misma medida.

La evaluación de rendimiento para copia de datos se realiza de manera muy similar a la escritura, con la salvedad de necesitar dos arrays de datos, uno del cual leer, y otro en el que depositar los datos. El diagrama explicativo es el siguiente:

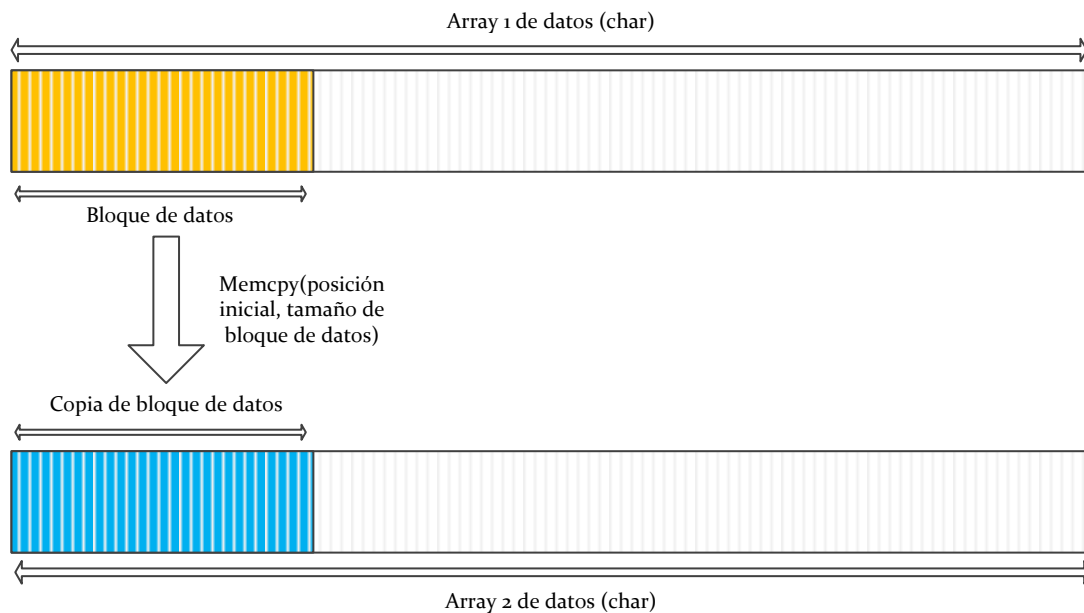


Ilustración 17: Funcionamiento de la evaluación de rendimiento de copia en memoria

Al igual que el módulo de escritura, en este módulo se evalúa la cantidad de datos que son copiados por segundo, midiéndose en MB/s.

Para estas dos pruebas, se definen distintos tamaños de bloque, de tal manera que resulta posible evaluar el rendimiento de los distintos niveles de caché de datos (en el caso de que se dispongan), y el rendimiento de la memoria RAM general.

Concretamente, el conjunto de datos total es de 8 MB, dado que la memoria se reserva estáticamente (hay arquitecturas que no disponen de unidades de memoria virtual, por lo que no se puede reservar dinámicamente), siendo subdividido en trozos de 2^n bytes. El tamaño de datos total es el adecuado para que sea utilizable en la gran mayoría de sistemas empotrados, al estar limitados en la cantidad de RAM disponible para la ejecución en programas.

Un ejemplo de ejecución es el siguiente (Ilustración 18), el PC utilizado contiene un procesador Intel Core 2 Duo con 32Kb de caché L1 de datos y 3Mb de caché L2:

```

~/vitiram
ViTi95@HP-HDX16-1160 ~/vitiram
$ gcc -O3 VitiRAM.c -o VitiRAM
ViTi95@HP-HDX16-1160 ~/vitiram
$ ./VitiRAM.exe
Benchmark de localidad temporal (Escritura)

Blocksize      Passes  Iterations      Result
8388608         1       1250      2410.21927 MB/s
4194304         2       1287      2640.00028 MB/s
2097152         4       5305      10504.94958 MB/s
1048576         8       5769      11378.70024 MB/s
524288         16      5834      11375.09258 MB/s
262144         32      5704      11564.11395 MB/s
131072         64      5708      11663.85932 MB/s
65536          128      5769      11336.77255 MB/s
32768          256      8402      16766.27422 MB/s
16384          512     8078      16502.55414 MB/s
8192           1024    7839      16079.98691 MB/s
4096           2048    7192      14636.48359 MB/s
2048           4096    6481      13136.05920 MB/s
1024           8192    5414      10844.26093 MB/s
512            16384   3750      7947.01987 MB/s
256            32768   2574      5155.73226 MB/s
128            65536   1959      3549.71859 MB/s
64             131072  1151      2270.21853 MB/s
32             262144  625      1256.91229 MB/s
16             524288  197      404.10320 MB/s
8              1048576  97      198.97399 MB/s
4              2097152  51      102.15326 MB/s
2              4194304  101     190.38628 MB/s
1              8388608  48      96.53103 MB/s

Benchmark de localidad temporal (Lectura + Escritura)

Blocksize      Passes  Iterations      Result
8388608         1       745      1552.89475 MB/s
4194304         2       754      1534.08149 MB/s
2097152         4       887      1697.19870 MB/s
1048576         8      2983      5502.42180 MB/s
524288         16     3088      6162.13369 MB/s
262144         32     3282      6422.70116 MB/s
131072         64     3088      6044.53054 MB/s
65536          128     3124      6282.55617 MB/s
32768          256     3124      6408.19610 MB/s
16384          512     7400      14538.29777 MB/s
8192           1024    7294      14497.37504 MB/s
4096           2048    6649      13422.16816 MB/s
2048           4096    6036      11858.53586 MB/s
1024           8192    4773      9451.50057 MB/s
512            16384   3456      6978.27057 MB/s
256            32768   2432      4833.79146 MB/s
128            65536   1662      3342.37921 MB/s
64             131072  1050      2234.04720 MB/s
32             262144  570      1174.04819 MB/s
16             524288  293      613.29249 MB/s
8              1048576  88      176.97372 MB/s
4              2097152  43      90.38333 MB/s
2              4194304  90      181.72583 MB/s
1              8388608  45      89.79785 MB/s

ViTi95@HP-HDX16-1160 ~/vitiram
$

```

Ilustración 18: Ejemplo de ejecución de VitiRAM

Para cada test, se puede observar cómo se comprueban las transferencias de datos con tamaños distintos de bloque, el número de pasadas ejecutadas con cada bloque y el número de iteraciones totales realizadas por cada conjunto de pasadas/bloque. En la columna más a la derecha, se muestra el ancho de banda en MB/s.

4.3 Método de trabajo

Aparte de especificar cómo son las pruebas de rendimiento necesarias para evaluar el soft-processor, es necesario especificar un método de trabajo con el que evolucionar el diseño de la FPGA y el procesador MicroBlaze.

Para realizar la aproximación a un diseño eficiente y con un buen rendimiento aplicaremos el método inductivo, realizando pequeñas progresiones e indagaciones por cada resultado conseguido

Fundamentalmente las iteraciones que se realizarán serán las siguientes:

1. Inicialmente, en la **primera iteración** se realizará el proceso iterativo con la placa de desarrollo Digilent Nexys3, probando las distintas características internas y módulos del procesador MicroBlaze. De las **distintas configuraciones hardware** que realicemos se elegirán varias de las mejores.
2. La **segunda iteración** se realizará comparando las plataformas de desarrollo Digilent Nexys3 y Digilent Atlys, probando las distintas características internas del procesador, pero centrándonos en los tamaños de **caché**, opciones de **memoria** y configuraciones del predictor de **saltos**. De las distintas pruebas realizadas elegiremos otra vez las mejores configuraciones hardware para pasarlas a la última iteración.
3. La **tercera iteración** se buscarán las mejores opciones para el compilador empleado por Xilinx, utilizando y comparando las diversas opciones de **optimización GCC**.
4. En la **cuarta** y última **iteración** compararemos las mejores configuraciones de la placa Digilent Atlys con **sistemas empotrados similares**, pero basados en otras arquitecturas. Estas arquitecturas son las descritas en capítulos anteriores, y analizaremos además del rendimiento la eficiencia energética de ellos.

En el siguiente capítulo se explicará cómo se han implementado los distintos benchmarks para las distintas arquitecturas, y se realizará el proceso iterativo de evolución y evaluación del rendimiento de los soft-processors MicroBlaze explicado anteriormente.

5 Implementación

En este apartado veremos cómo se ha de crear el entorno de trabajo y de pruebas para la evaluación del rendimiento en las distintas plataformas, además de ver cómo se ha de implementar propiamente las pruebas de rendimiento en las distintas arquitecturas.

Primeramente vamos a ver la implementación en las placas de desarrollo para el procesador MicroBlaze, seguido de otras arquitecturas que utilizaremos para comparar los resultados, concretamente las especificadas en el apartado 3.2.3 de este documento.

5.1 Digilent Nexys3 y Atlys

Las placas de desarrollo Digilent Nexys3 y Digilent Atlys son las siguientes:

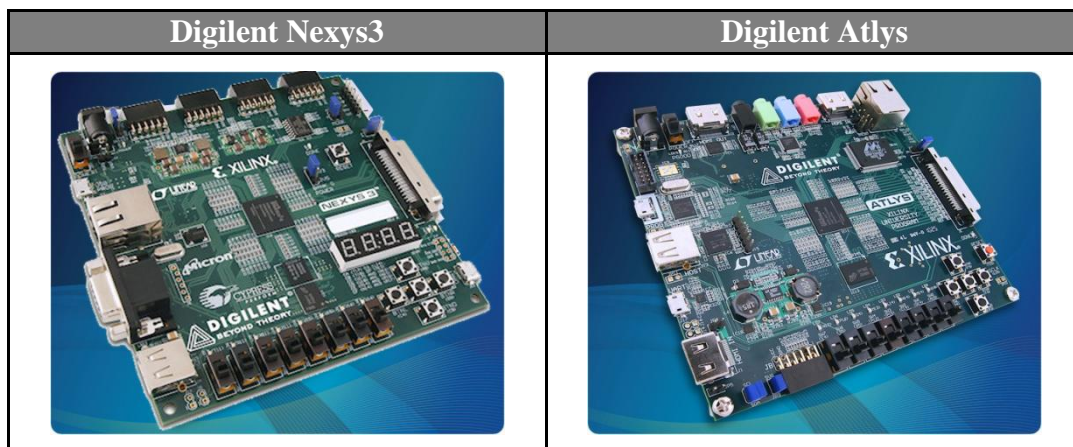


Tabla 27: Placas de desarrollo Digilent Nexys3 y Digilent Atlys

La creación de los sistemas empotrados basados en el procesador MicroBlaze y las FPGAs de Digilent empieza con la descarga de las librerías de soporte ofrecidas para las distintas placas. Sin estas librerías es imposible la implementación de cualquier tipo de hardware dentro de las FPGAs.

Con las librerías disponibles, ejecutamos el entorno de desarrollo hardware Xilinx Platform Studio, también conocido como XPS, el cual nos permitirá diseñar el sistema empotrado en la FPGA con el soft-processor a través del Base System Builder. El Base System Builder simplifica el proceso de implementación del procesador MicroBlaze, siguiendo una serie de pasos sencillos. La base del sistema será el bus PLB, puesto que es el tipo de bus recomendado por Digilent para sus FPGAs.

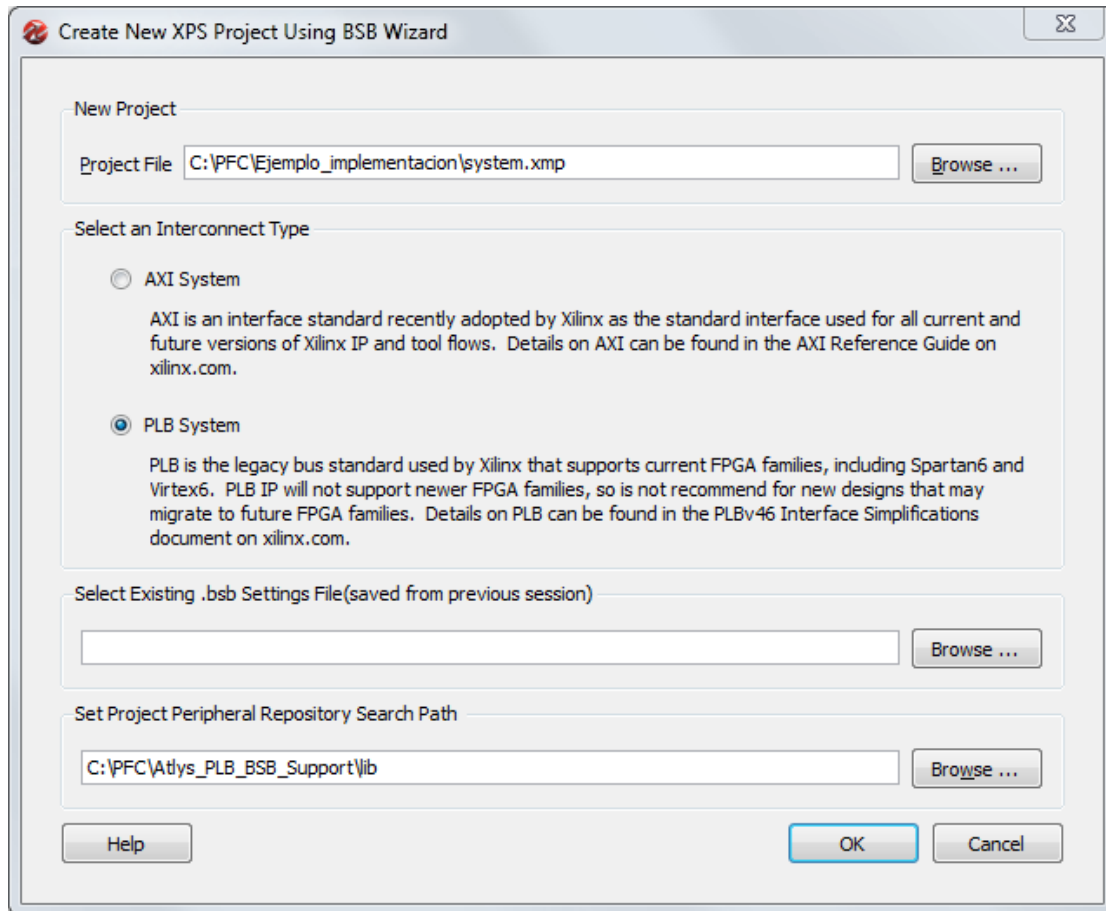


Ilustración 19: Base System Builder de Xilinx Platform Studio

En las características principales que se permiten escoger para el procesador MicroBlaze, optaremos por implementar un único MicroBlaze, puesto que las configuraciones multiprocesador de MicroBlaze no comparten la memoria RAM, sino que utilizan *mutex* o sistemas *mailbox* para la compartición de información entre los procesadores. Además, el sistema operativo base del MicroBlaze, el XilKernel, carece de soporte para multiproceso SMP, por lo que no resulta idónea su implementación.

Una vez elegido el número de procesadores MicroBlaze, se puede establecer la frecuencia base a la que funcionará el soft-processor, la opción de incluir una unidad FPU de coma flotante simple y la cantidad de memoria RAM local para el MicroBlaze.

Este es un ejemplo de dicha pantalla:

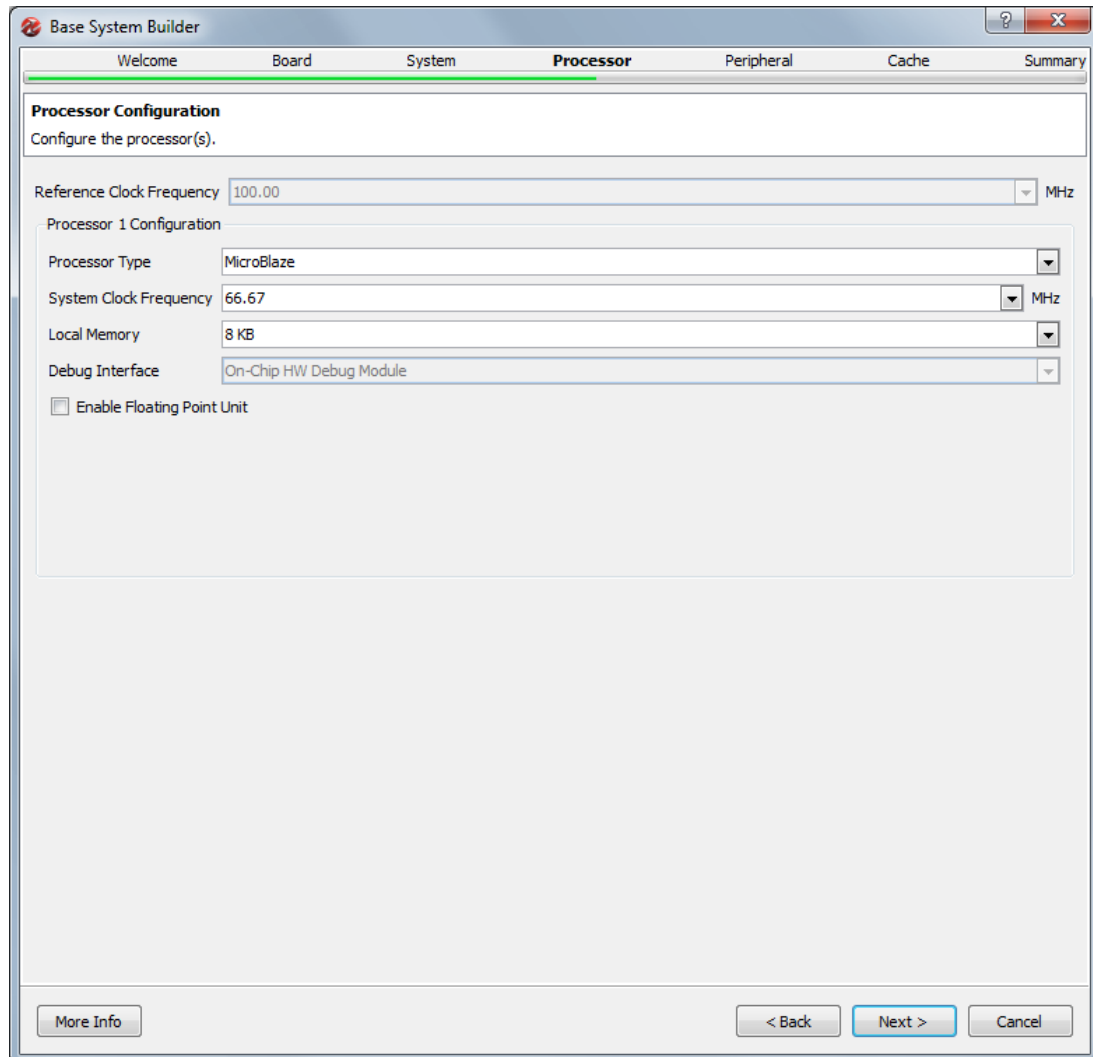


Ilustración 20: Parámetros básicos configurables del soft-processor MicroBlaze

De estos campos sólo modificaremos la frecuencia base del procesador, y la opción de la unidad FPU, puesto que la memoria local (BRAM) del procesador no será utilizada, al disponer módulos de memoria extendida fuera de la FPGA para ambas placas de desarrollo. Una vez escogidas estas opciones, se pueden establecer los módulos complementarios para el soporte de los distintos dispositivos, como pueden ser la memoria RAM externa, dispositivos de video, teclados, etc.

En estas opciones modificaremos la velocidad de transmisión de datos del puerto RS-232 de 9600 baudios a 115200, con el objetivo de no perder datos en la consola, y añadiremos un *timer* con el que podremos realizar las mediciones de tiempos necesarias (Ilustración 21).

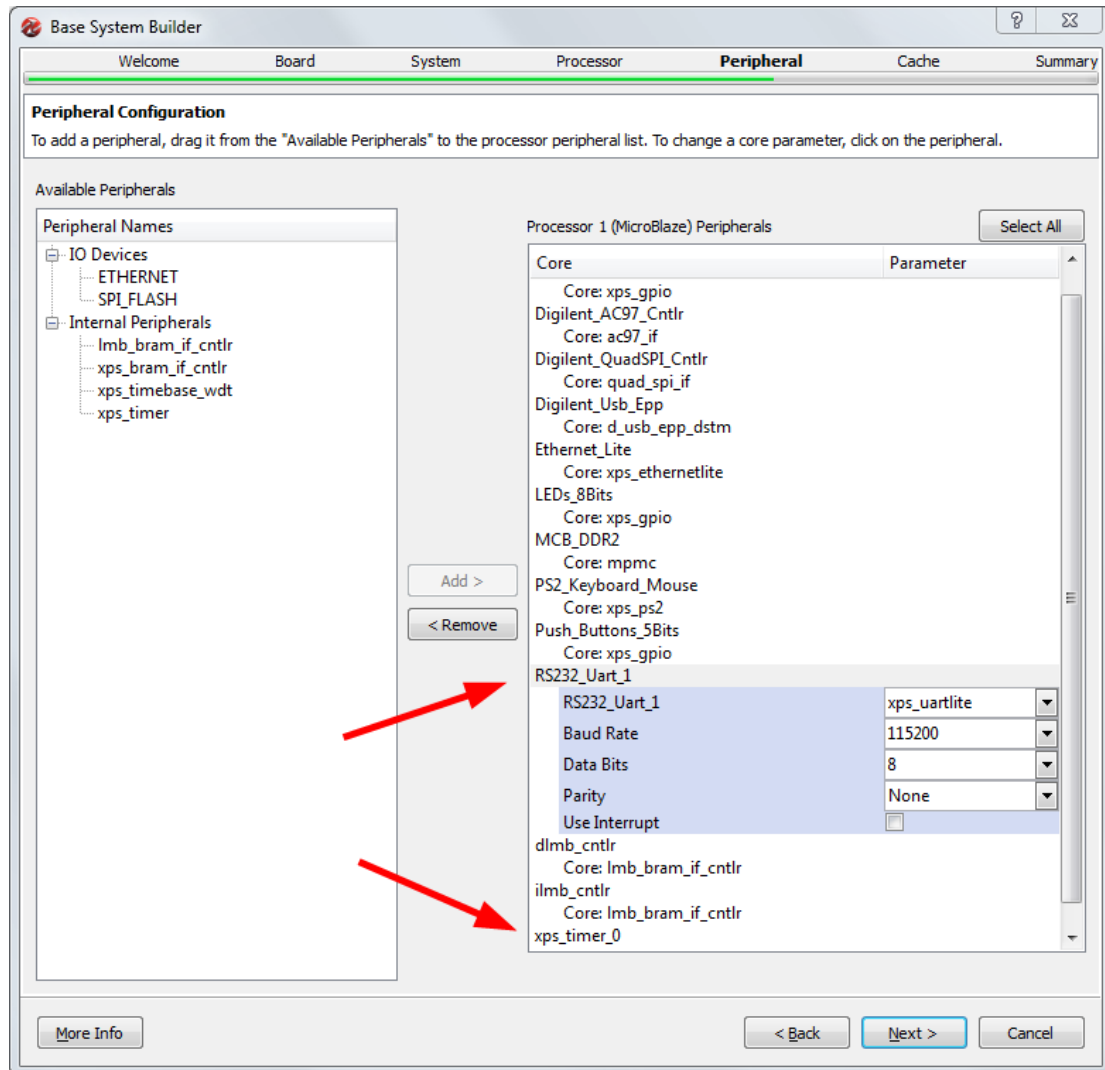


Ilustración 21: Configuración de dispositivos necesarios para la arquitectura MicroBlaze

Para finalizar el proceso de construcción de la base hardware, es necesario especificar si se desea utilizar cachés de instrucciones y de datos, cuánta cantidad se va a utilizar para cada una de ellas en el caso de activarla y sobre que dispositivo de memoria se va a utilizar.

En la Ilustración 22 se muestra un ejemplo de dicha ventana de configuración, para la placa de desarrollo Digilent Atlys:

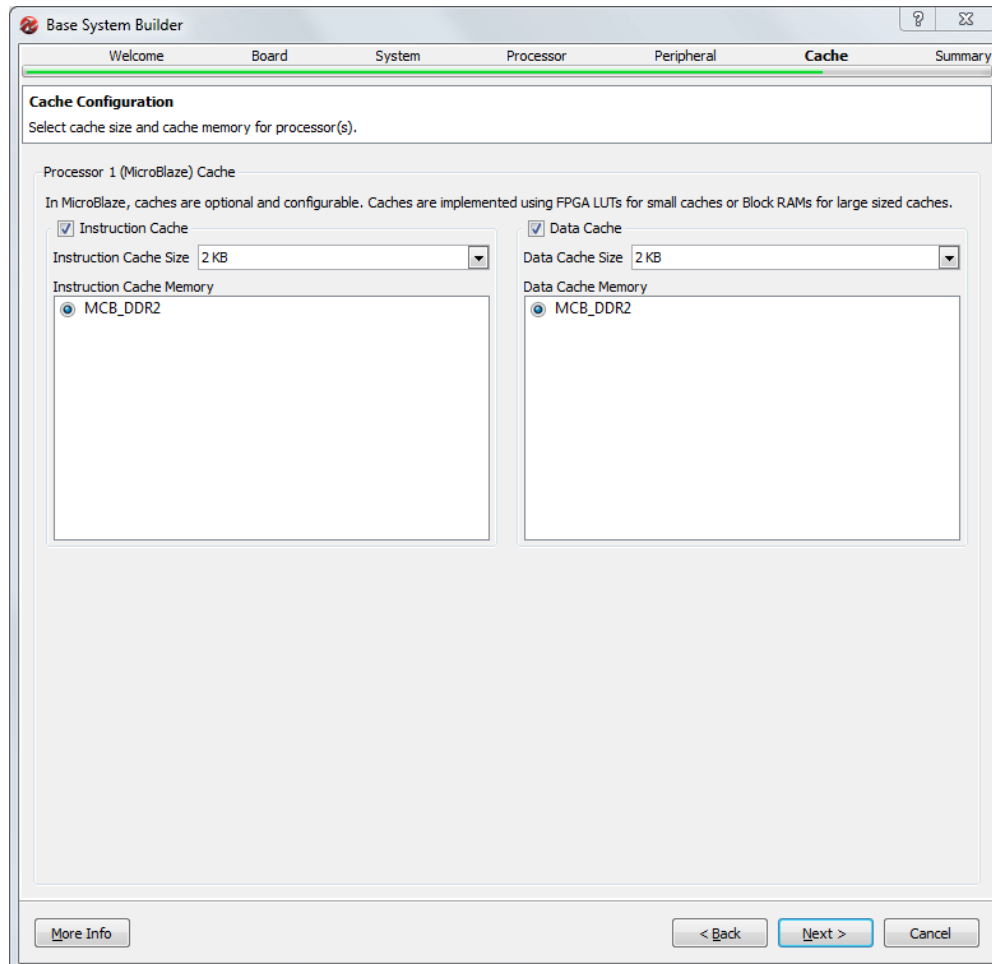


Ilustración 22: Configuración de memoria caché para la arquitectura MicroBlaze

Con el sistema base ya configurado, es posible realizar una configuración avanzada de las características del procesador MicroBlaze, seleccionando en el XPS dicho componente. Desde esta ventana se realizarán todas las modificaciones posibles con un sentido en el proceso iterativo para la obtención del mejor soft-processor, incluyendo cambios en los componentes internos y realizando cambios en las opciones de memoria.

La ventana de configuración avanzada es la siguiente:

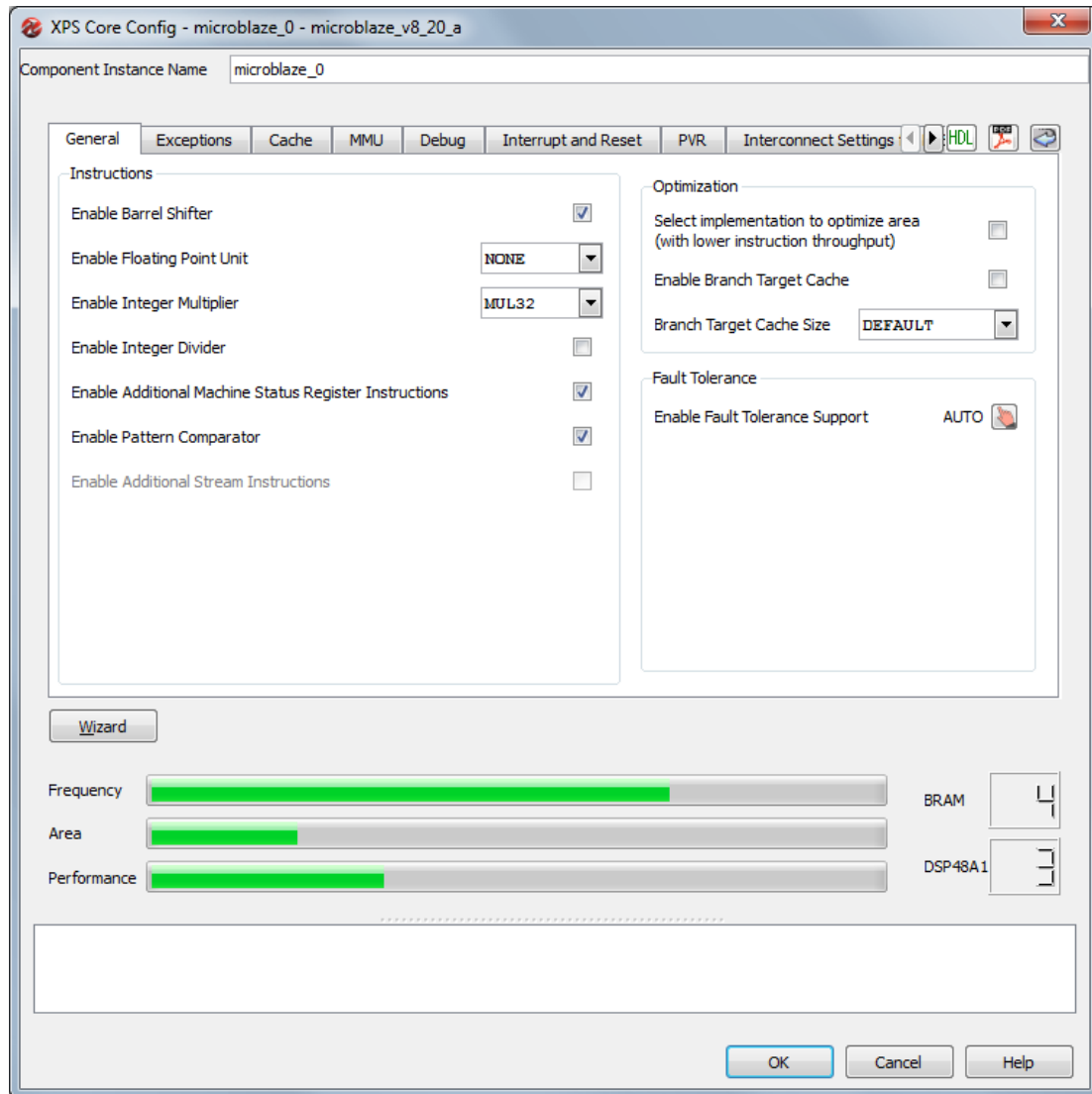


Ilustración 23: Opciones avanzadas del procesador MicroBlaze

Para finalizar el proceso de generación e implementación del hardware dentro de la FPGA, bastará con realizar la acción de generación del bitstream. El bitstream es el fichero de configuración final que determina la estructura que deberá tomar la FPGA internamente para el funcionamiento del hardware diseñado.

El proceso de generación del bitstream es lento y pesado, puesto que el proceso de compilación y síntesis de los diseños de hardware en el lenguaje VHDL es complejo, sobre todo a la hora de implementar el propio diseño en la FPGA. En el proceso iterativo se comprobó que las configuraciones más complejas, es decir, aquellas que constan de más componentes, tienden a alargar desmesuradamente los procesos de compilación.

Una vez se ha finalizado el proceso de generación del bitstream, es posible enlazar la parte hardware con la parte software. Para poder generar software para la arquitectura MicroBlaze, deberemos utilizar el entorno de desarrollo proporcionado por Xilinx denominado Xilinx Software Development Kit, también conocido como XSDK. Desde el XPS se realizará la acción de exportar el hardware creado al entorno de desarrollo, lo que abrirá automáticamente el XSDK y configurará lo necesario para su funcionamiento.

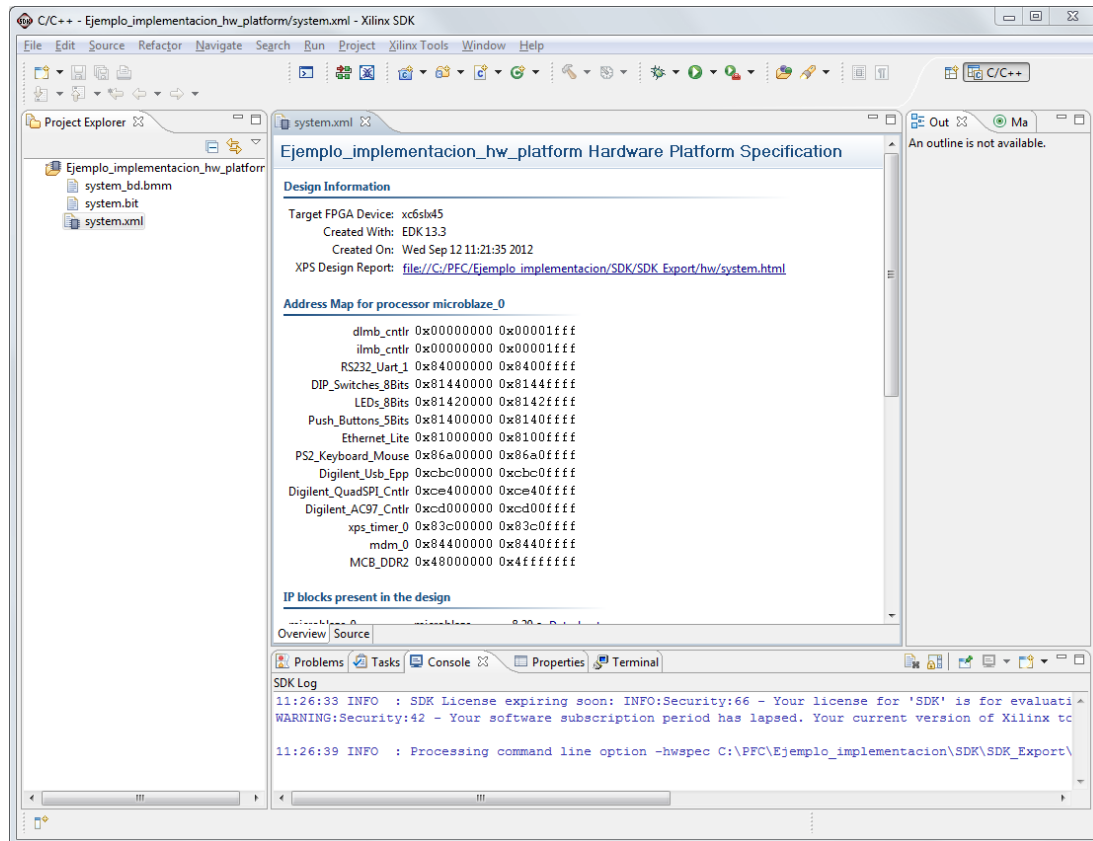


Ilustración 24: Entorno de desarrollo Xilinx Software Development Kit para el procesador MicroBlaze

La siguiente acción necesaria para poder desarrollar software en la arquitectura MicroBlaze, es establecer un sistema operativo base. El XSDK puede trabajar con dos modalidades, una denominada *standalone* en la que no existe un sistema operativo base (se ejecuta directamente el código compilado, junto con las librerías que se le adjunen), y otra denominada *XilKernel*, que implementa un sistema operativo mínimo basado en POSIX para sistemas empujados. Para este proyecto trabajaremos con el *XilKernel*, al ofrecer todo lo necesario para poder realizar las pruebas de rendimiento.

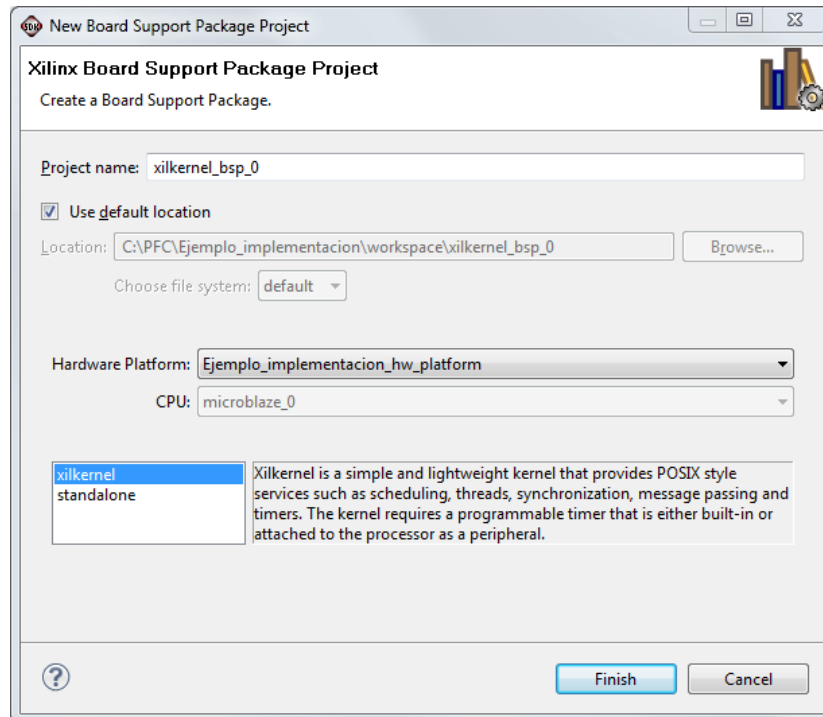


Ilustración 25: Ventana de selección de sistema base para la plataforma MicroBlaze

El XilKernel para su correcto funcionamiento es necesario modificarlo, para que sea posible la utilización del *timer* implementado en hardware. De no realizar esta acción, resulta imposible utilizar las funciones dedicadas a la obtención de tiempos, lo que imposibilitaría la implementación de cualquier tipo de benchmark. Para ello, en las opciones del XilKernel es necesario activar esta funcionalidad:

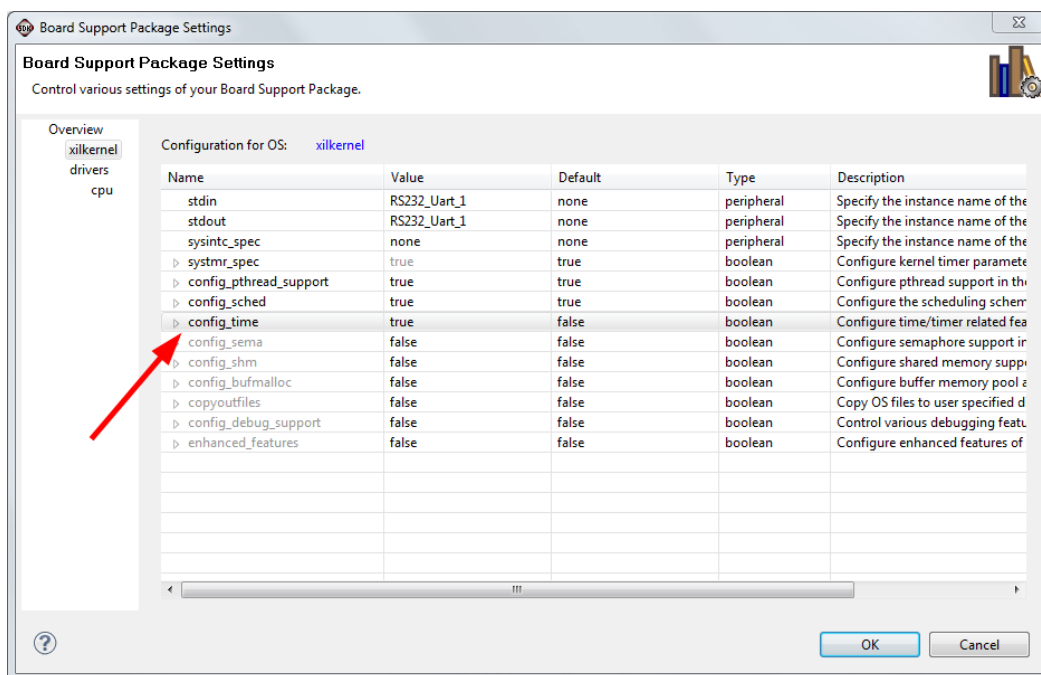


Ilustración 26: Configuración de los timers en el XilKernel

A partir de este punto ya se puede crear cualquier tipo de proyecto software para la plataforma hardware diseñada. Cabe destacar que no todas las funciones que existen en el lenguaje de programación C/C++ están implementadas, a pesar de utilizar el compilador GCC. Esto representa un serio problema, puesto que los programas creados podrían simplemente no funcionar.

En nuestro caso, la única problemática reside en que las funciones de tiempo de la librería “*time.h*” no están implementadas. Sin embargo, para este problema existe una fácil solución, y es utilizar las librerías que proporciona adicionalmente Xilinx para este cometido.

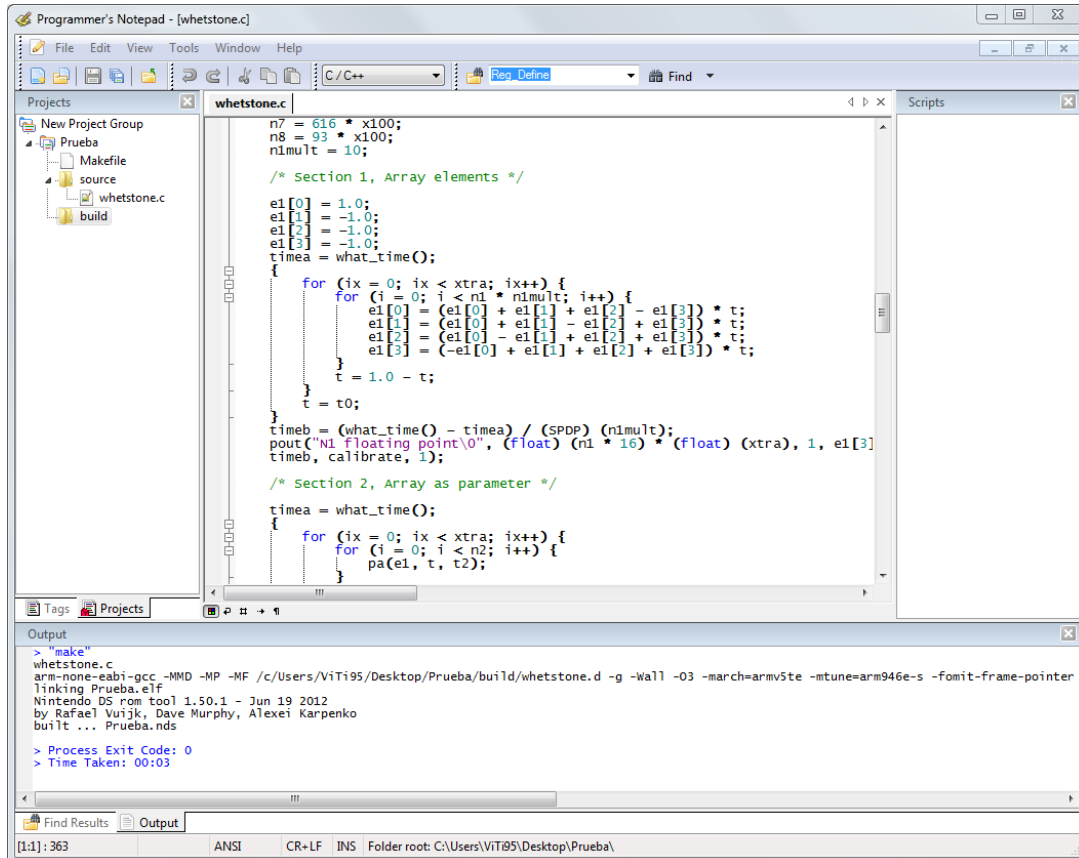
Otro punto a considerar es que el XilKernel no tiene mecanismos que permitan mostrar resultados por pantalla con el hardware que ofrece las placas de desarrollo de Digilent, por lo que la única manera de poder visualizar los resultados será a través del puerto USB mediante comunicación serie y un terminal en el ordenador (el propio entorno de desarrollo incluye uno, aunque también sería posible utilizar programas como *PuTTY* para Windows).

5.2 Nintendo DS

La implementación de las pruebas de rendimiento para la consola portátil Nintendo DS se realiza mediante el entorno de desarrollo *devKitPro* (concretamente la variante de *devKitARM*), que incluye todo el material necesario para crear ejecutables para esta videoconsola portátil^[47].

Todo el desarrollo del software se realiza desde el programa *Programmer's Notepad*, que está especializado en la creación de proyectos para *devKitPro*.

En la Ilustración 27 se puede apreciar un ejemplo de desarrollo software en dicho entorno de desarrollo:



```
Programmer's Notepad - [whetstone.c]
File Edit View Tools Window Help
C/C++
Reg. Define Find
Projects
New Project Group
Prueba
  Makefile
  source
  whetstone.c
  build
whetstone.c
n7 = 616 * x100;
n8 = 93 * x100;
nmult = 10;

/* Section 1, Array elements */
e1[0] = 1.0;
e1[1] = -1.0;
e1[2] = -1.0;
e1[3] = -1.0;
timea = what_time();
for (ix = 0; ix < xtra; ix++) {
  for (i = 0; i < nmult; i++) {
    e1[0] = (e1[0] + e1[1] + e1[2] - e1[3]) * t;
    e1[1] = (e1[0] + e1[1] - e1[2] + e1[3]) * t;
    e1[2] = (e1[0] - e1[1] + e1[2] + e1[3]) * t;
    e1[3] = (-e1[0] + e1[1] + e1[2] + e1[3]) * t;
  }
  t = 1.0 - t;
  t = t0;
}
timeb = (what_time() - timea) / (SPDP) (nmult);
pout("N1 floating point\n", (float) (n1 * 16) * (float) (xtra), 1, e1[3]);
timeb, calibrate, 1);

/* Section 2, Array as parameter */
timea = what_time();
for (ix = 0; ix < xtra; ix++) {
  for (i = 0; i < nmult; i++) {
    pa(e1, t, t2);
  }
}

Output
> "make"
whetstone.c
arm-none-eabi-gcc -MMD -MP -MF /c/Users/ViTi95/Desktop/Prueba/build/whetstone.d -g -Wall -O3 -march=armv5te -mtune=arm946e-s -fomit-frame-pointer
Linking Prueba.elf
Nintendo DS rom tool 1.50.1 - Jun 19 2012
by Rafael Vuijk, Dave Murphy, Alexei Karpenko
built ... Prueba.nds

> Process Exit Code: 0
> Time Taken: 00:03

Find Results
[1:1]: 363
ANSI CR+LF INS Folder root: C:\Users\ViTi95\Desktop\Prueba\
```

Ilustración 27: Ejemplo de proyecto en Programmer's Notepad para Nintendo DS

Por otro lado, la Nintendo DS carece de un sistema operativo base, por lo que es necesario utilizar una librería que implemente las funcionalidades que contempla el lenguaje C/C++ a través del compilador GCC. Para ello, utilizaremos la librería *libNDS*, que abarca una gran cantidad de funcionalidad, y simplifica la implementación y mostrado de resultados por pantalla.

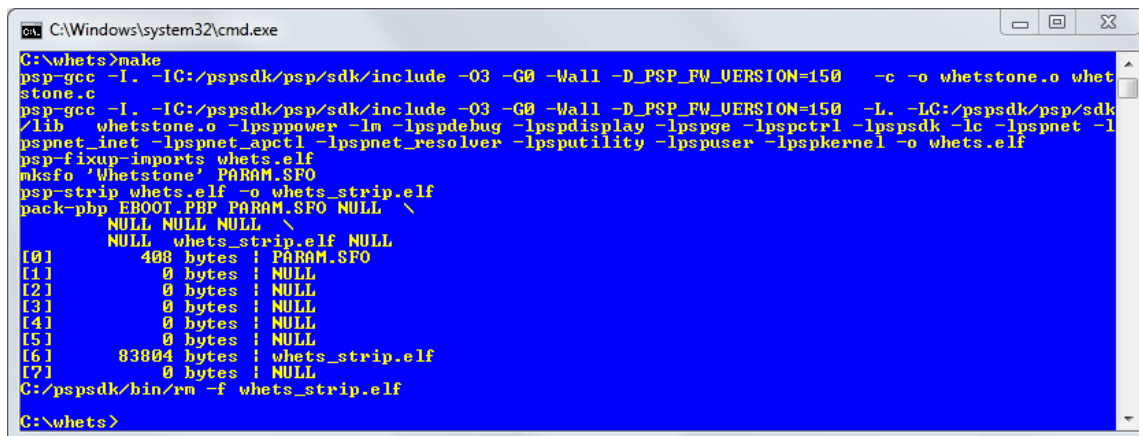
La única dificultad que entraña el desarrollo de los benchmarks para la plataforma de Nintendo es la no implementación de la librería *time.h* de C, por lo que se debe recurrir a programación a bajo nivel de los timers de la propia consola, y simular las funciones de tiempo con dichos timers. Además, es necesario añadir código especializado para poder devolver resultados por pantalla.

Finalmente, para poder realizar la evaluación de los benchmarks resulta necesaria la utilización de un *flashcart*, que permita cargar software no firmado para la consola, puesto que Nintendo restringe la ejecución de código directamente bajo sus plataformas.

5.3 Sony Playstation Portable

El desarrollo de software para la *Playstation Portable* se puede realizar desde el *devKitPro*, aunque en este caso se ha optado por la utilización del entorno de desarrollo *Minimalist PSPSDK*. Este SDK lleva integrado todo lo necesario para poder desarrollar y compilar software para esta plataforma, aunque no incluye un entorno IDE que simplifique la labor de desarrollo^[48].

Para la adaptación de los benchmarks, utilizamos el programa *Notepad++*, que facilita en cierta medida esta parte. La compilación se realiza a través de consola directamente, al carecer *Notepad++* la opción de ejecutar comandos de consola que automaticen el proceso. Este proceso de compilación se ayuda del sistema automatizado de compilación *make*.



```
C:\Windows\system32\cmd.exe
C:\whets>make
psp-gcc -I. -IC:/pspsdk/psp/sdk/include -O3 -G0 -Wall -D_PSP_FW_VERSION=150 -c -o whetstone.o whetstone.c
psp-gcc -I. -IC:/pspsdk/psp/sdk/include -O3 -G0 -Wall -D_PSP_FW_VERSION=150 -L. -LC:/pspsdk/psp/sdk/lib -o whetstone.o -lpsppower -lm -lpspdebug -lpspdisplay -lpspge -lpspctrl -lpspsdk -lc -lpspnet -lpspnet_inet -lpspnet_apctl -lpspnet_resolver -lpsputility -lpspuser -lpspkernel -o whets.elf
psp-fixup-imports whets.elf
mksfo 'Whetstone' PARAM.SFO
psp-strip whets.elf -o whets_strip.elf
pack-pbp EBOOT.PBP PARAM.SFO NULL \
NULL NULL NULL \
NULL whets_strip.elf NULL
[0] 408 bytes : PARAM.SFO
[1] 0 bytes : NULL
[2] 0 bytes : NULL
[3] 0 bytes : NULL
[4] 0 bytes : NULL
[5] 0 bytes : NULL
[6] 83804 bytes : whets_strip.elf
[7] 0 bytes : NULL
C:/pspsdk/bin/rm -f whets_strip.elf
C:\whets>
```

Ilustración 28: Ejemplo de compilación de proyecto para Sony Playstation Portable

Otra opción hubiese sido utilizar el entorno de desarrollo Eclipse adaptado para C/C++, y adaptar su configuración para utilizar los ejecutables de compilación GCC del SDK. Sin embargo, esta opción hubiese requerido mucho más tiempo de desarrollo y no aportaría mucho a la realización de los benchmarks.

El desarrollo de las adaptaciones de los benchmarks a esta plataforma no ha requerido prácticamente cambios sustanciales, dado que la única diferencia respecto al GCC que implementan los ordenadores actuales es que las funciones de devolución de resultados por pantalla no están implementadas, aunque sí existen alternativas a esta problemática. El SDK contempla funciones específicas de impresión de resultados por pantalla, que funcionan exactamente igual que las ofrecidas por C, por lo que solamente es necesario establecer un renombrado de las funciones *printf* a las ofrecidas por el SDK.

Finalmente, para la ejecución de código en esta plataforma es necesario disponer de una Playstation Portable modificada a nivel software, para que pueda ejecutar código no firmado nativamente, dado que Sony imposibilita al igual que Nintendo esta posibilidad.

5.4 Linux, Microsoft DOS y Microsoft Windows NT

La implementación para los sistemas operativos de ordenadores con arquitectura x86 o x86-64 es la más sencilla de los cuatro. Los compiladores para estas plataformas contemplan al 100% el estándar descrito por C en los compiladores GCC y los derivados de Watcom.

El compilador GCC permite crear código directamente ejecutable en los sistemas operativos Linux, Mac OS X, y en los sistemas operativos de Microsoft basados en el kernel NT. Para esta última plataforma es necesario tener instalado un entorno POSIX que permita compilar y ensamblar ejecutables, y esto se realiza a través del software Cygwin.

Este compilador es la base para todos los demás, por lo que todas las pruebas de rendimiento analizadas y diseñadas se programan para este entorno. Cabe destacar que no existe un IDE único y exclusivo para realizar proyectos con este compilador, por lo que se ha decidido, al igual que en la implementación para Playstation Portable, utilizar el programa Notepad++, y compilar a través de consola y la utilidad *make*.

Por otro lado, para la compilación de las pruebas de rendimiento en sistemas operativos más antiguos, como puedan ser Microsoft DOS y derivados, es necesaria la utilización de compiladores derivados de Watcom. Concretamente, para este proyecto se utiliza la versión libre del mismo, denominada OpenWatcom, y que permite realizar ejecutables para estos sistemas operativos, además de poder generar ejecutables para los actuales NT.

OpenWatcom integra un IDE completo y especializado para el desarrollo de software, por lo que la compilación del código generado para GCC resulta fácil y sencilla.

La Ilustración 29 un ejemplo de la compilación de un proyecto en OpenWatcom:

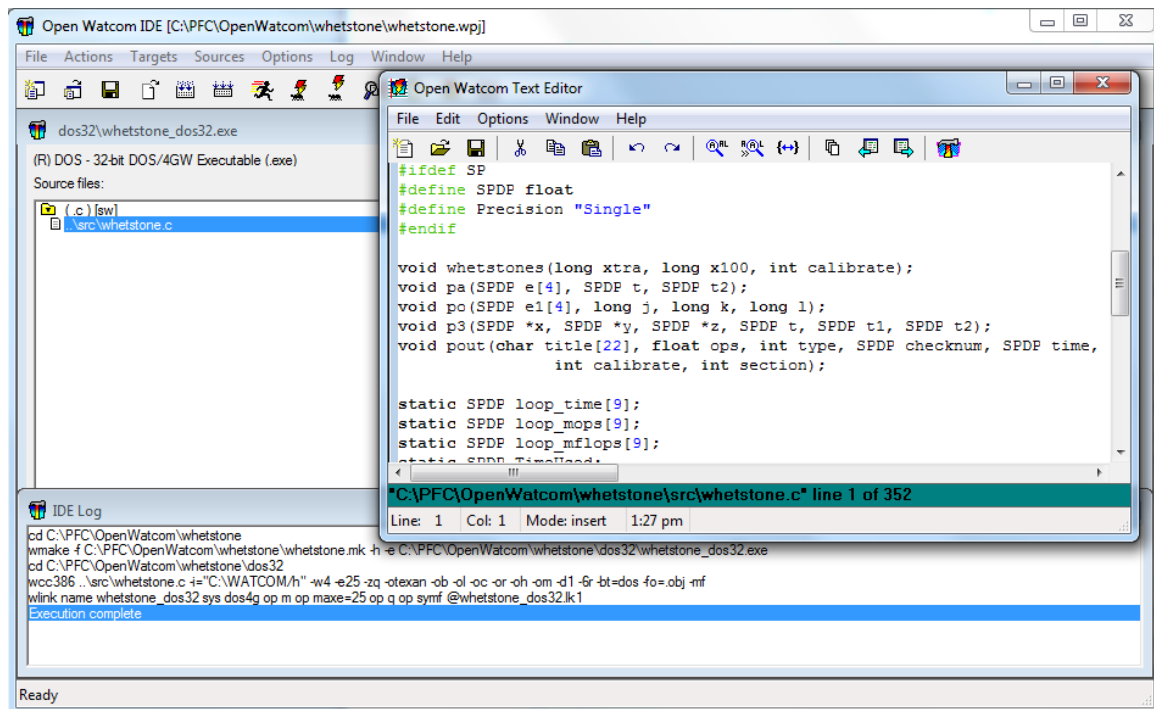


Ilustración 29: Ejemplo de compilación de proyecto en OpenWatcom

Cabe destacarse que OpenWatcom también es capaz de generar código para sistemas operativos superiores como puedan ser Linux o Windows, pero en estos casos es preferible el uso de GCC al ser más nuevo, y tener mejor compatibilidad con procesadores más actuales.



6 Evaluación y análisis de resultados

Una vez tenemos implementados los distintos benchmarks, podemos comenzar el proceso iterativo de evaluación de los rendimientos bajo las distintas configuraciones de FPGAs. En la primera iteración usaremos únicamente la placa de desarrollo Digilent Nexys3.

6.1 Primera iteración

Tal y como dijimos anteriormente, empezaremos utilizando la placa de desarrollo Nexys3 con todas las opciones y módulos del MicroBlaze desactivados, a una frecuencia de 66 MHz y con las cachés desactivadas. La única optimización activada es el uso de un pipeline de 5 etapas en vez de uno de 3.

Esta es en teoría la configuración mínima que debe ofrecer el peor rendimiento posible. Desde esta configuración iremos evolucionando la arquitectura y activando los módulos que ofrezcan un mejor rendimiento, en términos de rendimiento ALU y en coma flotante (Dhrystone, Whetstone y Linpack).

En el caso de activar un módulo y ofrecer un peor rendimiento, dicho módulo se desactivará y se omitirá en el proceso iterativo. Esto será debidamente justificado en cada iteración.

La primera fase iterativa trabajará con los siguientes aspectos de la arquitectura^[8]:

- **Barrel Shifter:** Es una unidad que permite desplazar los bits de una determinada palabra, incrementa el rendimiento del procesador a cambio de incrementar el espacio usado en la FPGA.
- **Floating Point Unit:** Es la unidad de coma flotante simple, permite realizar operaciones del conjunto IEEE-754. Las operaciones básicas de esta FPU incluyen suma, resta, multiplicación, división y comparación. Existe la opción de extender la FPU y añadir instrucciones para la realización de raíces cuadradas, conversión de coma flotante a entero y viceversa. Incrementa el rendimiento en coma flotante simple, a cambio de incrementar en gran medida el espacio ocupado.
- **Integer Multiplier:** Unidad de multiplicado de enteros, de 32 o 64 bits. Activar esta unidad no requiere prácticamente espacio dentro de la FPGA.
- **Integer Divider:** Unidad de división de enteros de 32 bits, incrementa el espacio del soft-processor a cambio de un aumento de rendimiento en determinadas situaciones.

- **Additional Machine Status Register Instructions:** Esta opción agrega dos estados nuevos en el MSR, esta opción agrega rendimiento a la hora de establecer los bits del MSR.
- **Pattern Comparator:** Es la unidad hardware que permite realizar comparaciones de datos a nivel de bit, búsquedas de datos y conteo de ceros al final de una trama de datos.
- **Branch Target Caché:** Unidad de predicción de saltos. Permite que los saltos inmediatos o instrucciones de tipo *return* bien predichos se ejecutan sin *overhead*. Cuando se produce un fallo de predicción de salto, el tiempo de recuperación es de dos ciclos de reloj. Incrementa el tamaño del MicroBlaze considerablemente.
- **Data Caché:** Sección del procesador dedicada al almacenaje de datos que se encuentran fuera del rango determinado por el LMB. Dispone de mapeo directo de 1 vía, y con posibilidad de alternar entre los modelos write-through y write-back.
- **Instruction Caché:** Sección del procesador dedicada al almacenaje de instrucciones que se encuentran fuera del rango determinado por el LMB, y con mapeo directo de 1 vía. La estructura que sigue la caché de instrucciones es la siguiente:

Las versiones de las distintas placas que son explicadas en esta iteración y en las siguientes quedarán todas recogidas dentro del Anexo B: Resumen de arquitecturas implementadas, con el objetivo de tener una visión unificada de las características que disponen.

Con la primera versión denominada **Nexys3 v001** y todas las unidades anteriores desactivadas, los resultados obtenidos para los benchmarks fueron los siguientes:

Prueba de rendimiento	Nexys3 v001
Dhrystone 2.1	4.15 DMIPS
Linpack SP Unroll	Sobrepasado tiempo de cómputo
Linpack SP Roll	Sobrepasado tiempo de cómputo
Linpack DP Unroll	Sobrepasado tiempo de cómputo
Linpack DP Roll	Sobrepasado tiempo de cómputo
Whetstone SP	Sobrepasado tiempo de cómputo
Whetstone DP	Sobrepasado tiempo de cómputo

Tabla 28: Resultados Nexys3 v001

El término “Sobrepasado tiempo de cómputo” determina que el tiempo necesitado para ejecutar el benchmark ha sobrepasado el tiempo máximo estipulado para poder obtener un resultado coherente. Es por esto que podemos suponer que el rendimiento obtenido tanto en Linpack como en Whetstone es prácticamente nulo.

Este resultado es completamente normal, puesto que los benchmarks Linpack y Whetstone están diseñados para poder evaluar el rendimiento en coma flotante, y por tanto, si esta unidad se encuentra desactivada, todas los cálculos en coma flotante han de ser simulados por el procesador, lo que implica un resultado mucho peor que si se tuviera una unidad especializada para ello.

Comparando el resultado obtenido en Dhrystone 2.1 con otras arquitecturas (concretamente de los resultados obtenidos por el propio Roy Longbottom), indican que el resultado obtenido es francamente malo, concretamente, se sitúa alrededor de procesadores de menor frecuencia y arquitectura desactualizada.

Los procesadores a los que nos referimos son los Intel 386SX (bus de datos de 16 bits en vez de 32 bits) a una frecuencia de 25MHz, que obtiene un resultado de 3.5 DMIPS y un AMD 386DX a una frecuencia de 40MHz con un resultado de 13.7 DMIPS. Hemos de recordar que la frecuencia a la que funciona la FPGA es de 66 MHz, y por ello consideramos que el resultado es malo.

En versión siguiente, la **Nexys3 v002**, se ha activado la unidad de desplazamiento de bits, lo que ha llevado a que Dhrystone 2.1 devuelva un resultado de 4.51 DMIPS. Este resultado implica que el rendimiento se ha incrementado en un 8%.

Comparando la ocupación del espacio dentro de la FPGA, el tamaño se ha visto incrementado de la siguiente manera:

Elemento	Nexys3 v001	Nexys3 v002
Biestables	5991	6032 (+0.68%)
LUTs	6838	6976 (+2%)
BRAM	8	8

Tabla 29: Ocupación de la FPGA en Nexys3 v001 y v002

Para la versión **Nexys3 v003** los bits adicionales para el control del MSR se activaron, obteniéndose los mismos resultados que en la anterior iteración. Esto indica que dicha optimización no representa una mejora real en el benchmark Dhrystone 2.1. En cuanto al espacio utilizado este se incremento en biestables y LUTs, sin llegar a representar más de un 0.5% cada uno.

La siguiente versión, la **v004**, supuso la activación del comparador de patrones a nivel de bit, obteniéndose un mejor resultado en el benchmark Dhrystone 2.1, con un resultado de 5.36 DMIPS. Esto supone una mejora del 18% respecto a las versiones 2 y 3. En cuanto el espacio añadido, el incremento de del espacio de LUTs solo representó un 1%.

Como se ha podido observar, los conjuntos de instrucciones añadidos que se centran en el procesamiento de datos han sido los que han conseguido incrementar la potencia de la arquitectura, por lo que en las siguientes versiones del soft-processor implementado en la Nexys3 se van a potenciar las unidades de procesamiento de datos. Concretamente, se han activado las siguientes unidades, una por cada distinta versión:

Versión de FPGA	Unidad de multiplicado de enteros	Unidad de división de enteros
Nexys3 v005	Multiplicador de 32 bits	Desactivada
Nexys3 v006	Multiplicador de 32 bits	Divisor de 32 bits
Nexys3 v007	Multiplicador de 64 bits	Divisor de 32 bits

Tabla 30: Unidades de multiplicación y división de enteros en Nexys3 v005, v006 y v007

Con estas versiones se espera mejorar el rendimiento obtenido en el benchmark Dhrystone 2.1, y se espera al menos conseguir algún resultado en los que utilicen coma flotante, puesto que estas unidades pueden acelerar dichos procesos.

Los resultados obtenidos demostraron que la optimización sólo se produjo al activar la unidad de multiplicación de enteros de 32 bits y la unidad de multiplicación de enteros de 64 bits. En ninguna de las pruebas realizadas la unidad de división de enteros resultó utilizada, por lo que su implementación se ha considerado poco efectiva. Este suceso es debido a que el compilador GCC es lo suficientemente inteligente como para optimizar las operaciones de división en otras más efectivas (esto se verificó analizando el código en ensamblador una vez compilado). Estos son los resultados obtenidos para las versiones 5, 6 y 7:

Prueba de rendimiento	Resultado Nexys3 v005	Resultado Nexys3 v006	Resultado Nexys3 v007
Dhrystone 2.1	5.41 DMIPS	5.41 DMIPS	5.41 DMIPS
Linpack SP Unroll	0.01 MFlops	0.01 MFlops	0.02 MFlops

Tabla 31: Resultados para Nexys3 v005, v006 y v007

La mejora de rendimiento obtenida en el benchmark Dhrystone 2.1 no es muy sustancial, cercana al 1%. Sin embargo, por primera vez el benchmark Linpack en su versión de coma flotante simple y con las secciones de código desenrolladas (optimización manual) ha conseguido devolver un resultado. El resultado obtenido para Linpack es malo, aunque la emulación de la unidad de coma flotante se haya visto mejorada al incluir instrucciones matemáticas que faciliten esta tarea.

Para hacernos una idea del resultado obtenido, un procesador AMD 386DX a 40MHz es capaz de obtener un resultado de 0.53 MFlops (53 veces más), incluso sin incluir una unidad de coma flotante.

Podemos concluir que la integración de dichas unidades es beneficioso, en especial la unidad de multiplicación para obtener una mejor eficiencia en la arquitectura. Ahora vamos a comparar la utilización de la FPGA en las tres versiones creadas:

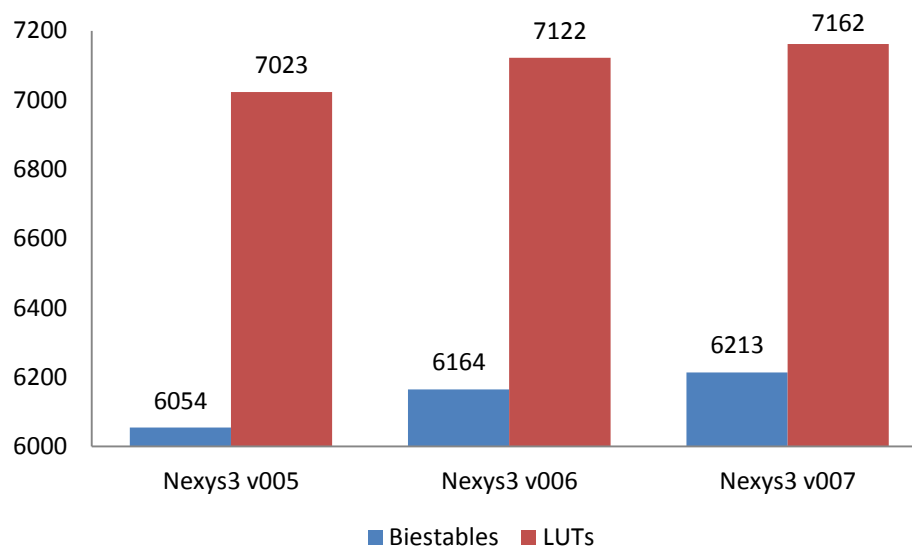


Ilustración 30: Ocupación en la FPGA en Nexys3 v005, v006 y v007

El incremento en el uso de la FPGA se ha visto afectado en un 2% para los LUTs y un 3% para los biestables, entre las versiones 4 y 7. Este incremento en el espacio no supone un gran inconveniente, puesto que el espacio utilizado en la FPGA Spartan6 LX16 no supera el 61% del total.

El siguiente paso consiste en probar la eficiencia del predictor de saltos. Para ello, en la **Nexys3 v008** se ha implementado dicha caché con el tamaño predeterminado por Xilinx. Concretamente, la implementación por defecto activa una caché de 256 direcciones de salto.

Esta implementación sorprendentemente no mejoró los resultados de los benchmarks ejecutados, aunque tampoco lo empeoró. La única diferencia fue un incremento en el tamaño del diseño, incluyendo la reserva de un módulo de memoria BRAM extra para la caché de saltos. Además, el número de biestables y LUTs utilizados se ha incrementado en un total aproximado del 2%.

El siguiente paso tomado es la inclusión de una unidad de coma flotante, con el objetivo de incrementar el rendimiento de las operaciones en coma flotante de precisión simple, y ver si es posible ejecutar los benchmarks Linpack y Whetstone con un mejor resultado.

Las implementaciones de la **unidad de coma flotante** se realizaron en los diseños de **Nexys3 v009 y v010**. La primera incluye la versión básica de la FPU, y la segunda incluye la versión extendida. Con estos dos diseños sí fue posible la ejecución correcta de todas las versiones de Linpack y Whetstone, tal y como vamos a ver ahora:

Prueba de rendimiento	Resultado de Nexys3 v009	Resultado de Nexys3 v010
Dhrystone 2.1	5.41 DMIPS	5.41 DMIPS
Linpack SP Unroll	0.79 MFlops	0.79 MFlops
Linpack SP Roll	0.55 MFlops	0.55 MFlops
Linpack DP Unroll	0.01 MFlops	0.01 MFlops
Linpack DP Roll	0.01 MFlops	0.01 MFlops
Whetstone SP	0.064 MWIPS	0.064 MWIPS
Whetstone DP	0.045 MWIPS	0.045 MWIPS

Tabla 32: Resultados para Nexys3 v009 y v010

Como era de suponer, el rendimiento no se ha mejorado en el benchmark Dhrystone 2.1, y si ha aumentado sustancialmente para los benchmarks Linpack y Whetstone en sus versiones de precisión simple. También se puede contemplar como la optimización manual de los bucles iterativos del benchmark Linpack suponen una mejora respecto a las versiones sin optimización.

Los resultados obtenidos sin embargo no son nada buenos comparadas con otras arquitecturas, por ejemplo, los 0.064 MWIPS conseguidos para Whetstone en coma flotante de precisión simple son comparables a los resultados obtenidos por ordenadores diseñados en los años 70 y 80, como por ejemplo el Borroughs B5500 (mismo resultado), las distintas versiones del PDP 11 (de 0.0027 a 0.714 MWIPS dependiendo del modelo) y las distintas versiones de IBM System/360 (de 0.0155 a 5 MWIPS dependiendo del modelo).

Además, se puede observar que el rendimiento en coma flotante de doble precisión es aún más pobre que en precisión simple, esto es ocasionado por la ausencia de soporte de precisión doble en las operaciones en coma flotante en la FPU integrada.

La inclusión de las operaciones extendidas en la FPU no ha supuesto mejora alguna en la arquitectura. Esto se puede deber a la poca representación de las instrucciones extra añadidas en los benchmarks Whetstone y Linpack, por lo que en las futuras versiones se utilizará la versión extendida de la FPU, ya que no empeora el resultado pero puede ser útil para aplicaciones que utilicen estas instrucciones extra.

En cuanto a la ocupación del procesador del MicroBlaze, ésta se incrementa considerablemente puesto que la unidad FPU, aunque esté optimizada en tamaño, necesita gran cantidad de biestables y LUTs para su correcta implementación. Aquí se puede observar la progresión en el uso de la FPGA en las versiones 9 y 10 respecto a la versión 8:

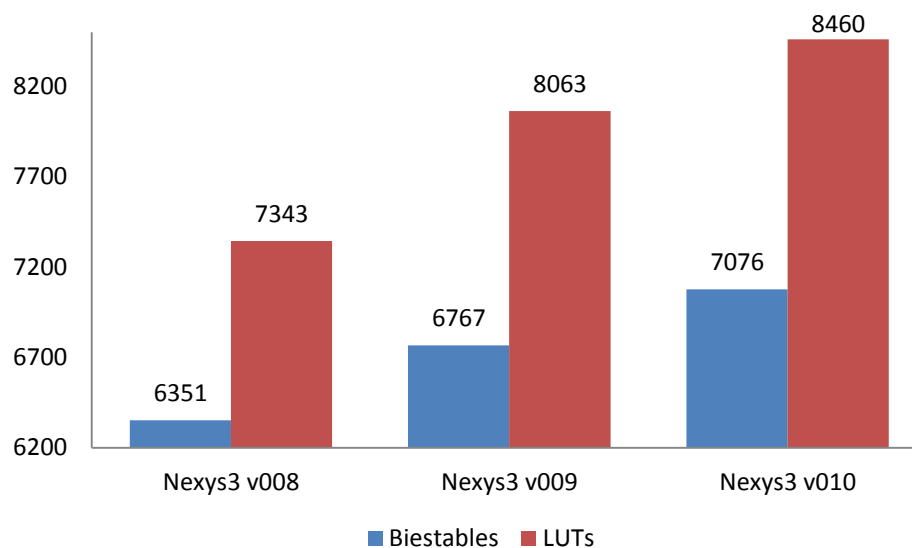


Ilustración 31: Ocupación de la FPGA en Nexys3 v008, v009 y v010

La utilización de la FPGA en los biestables se incrementa en un 6% para la versión 9, y un 11% en la versión 10 respecto a la versión 8. Para los LUTs ocurre algo similar, incrementándose un 9% y un 15% respectivamente. Se podría considerar que este incremento en el espacio utilizado es un impedimento, pero considerando que el rendimiento en las operaciones en coma flotante se ha incrementando hasta en un 79%, se puede considerar que es completamente necesario.

Después de probar todos los módulos que se pueden agregar a la arquitectura MicroBlaze, hemos podido comprobar que los resultados no son todo lo buenos que se esperaban. Esto ha ocurrido en gran medida por la falta de implementación de memorias caché para instrucciones y para datos, dado que éstas incrementan sustancialmente el rendimiento en todas las arquitecturas al acelerar el proceso de lectura/escritura de información desde la memoria RAM (ésta es mucho más lenta que los bloques BRAM que implementa la FPGA). Es por ello que en las siguientes versiones del soft-processor se implementará **memoria caché** para instrucciones y para datos, en pequeños incrementos para comprobar además la escalabilidad de la misma.

Las versiones **Nexys3 v011 a v019** son las diseñadas con implementación de caché, desde los 128 bytes (64 bytes para instrucciones y 64 bytes para datos) hasta los 32 KBytes (16 KBytes para instrucciones y 16 KBytes para datos). Cabe destacar que la arquitectura MicroBlaze soporta tamaños de caché mayores que 32 KBytes, pero las limitaciones de BRAM de la FPGA Spartan6 LX16 no permiten usar estas configuraciones.

Los resultados obtenidos fueron los siguientes (para su correcta visualización, se ha añadido el tamaño total de caché de datos e instrucciones asociadas al MicroBlaze):

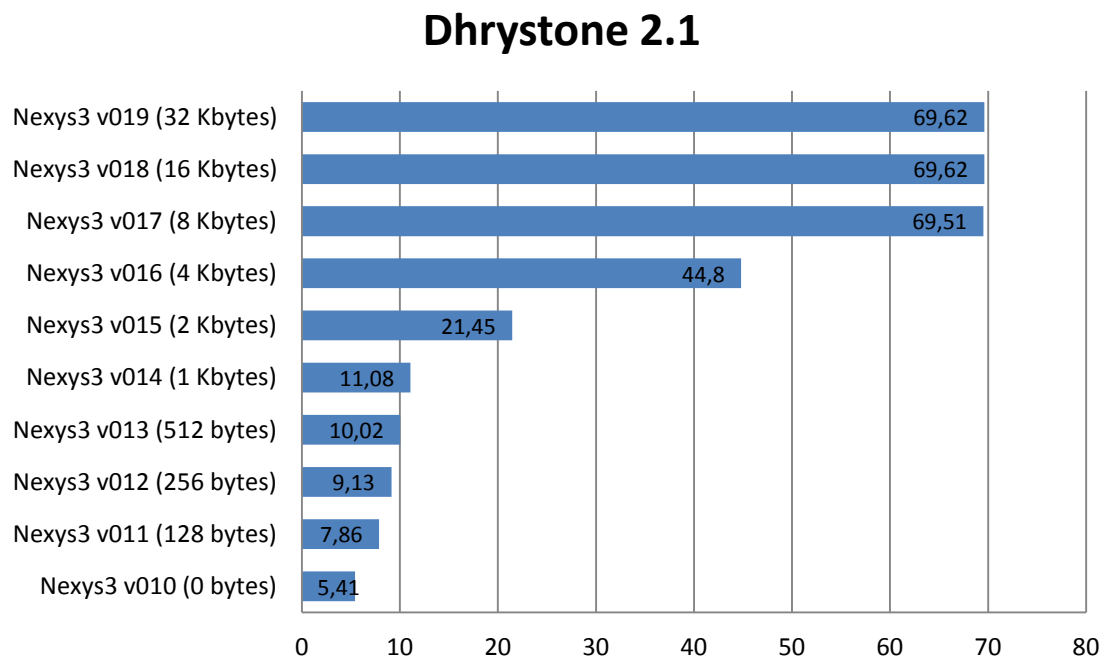


Ilustración 32: Benchmark Dhrystone 2.1 para comparativa de tamaños de caché en Nexys3

Linpack de precisión simple

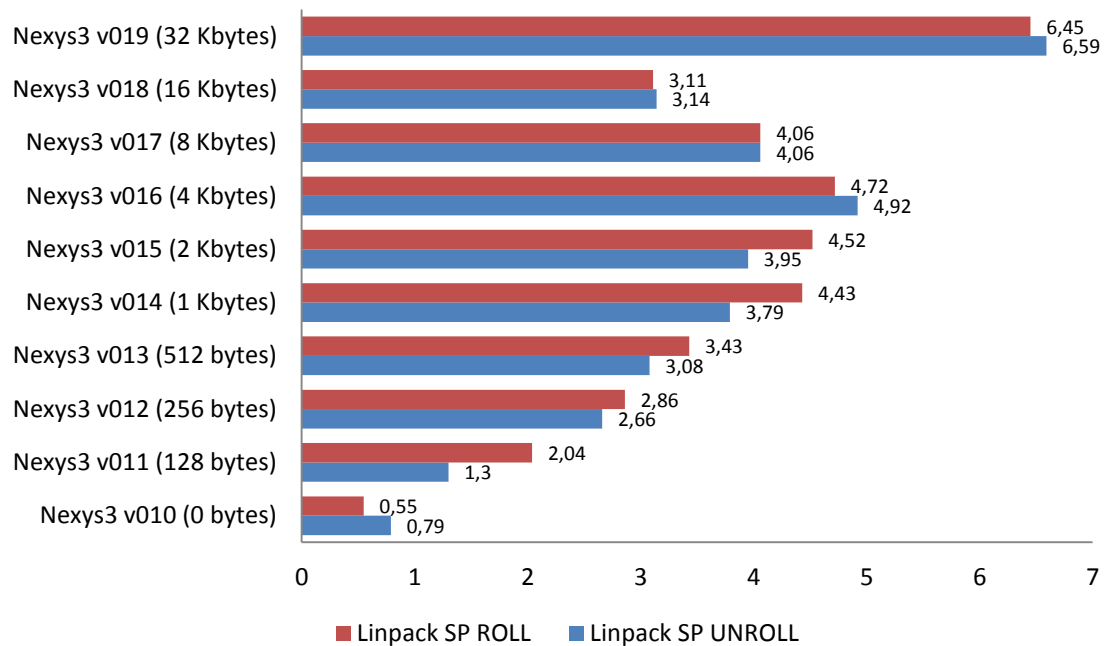


Ilustración 33: Benchmark Linpack para comparativa de tamaños de caché en Nexys3

Whetstone

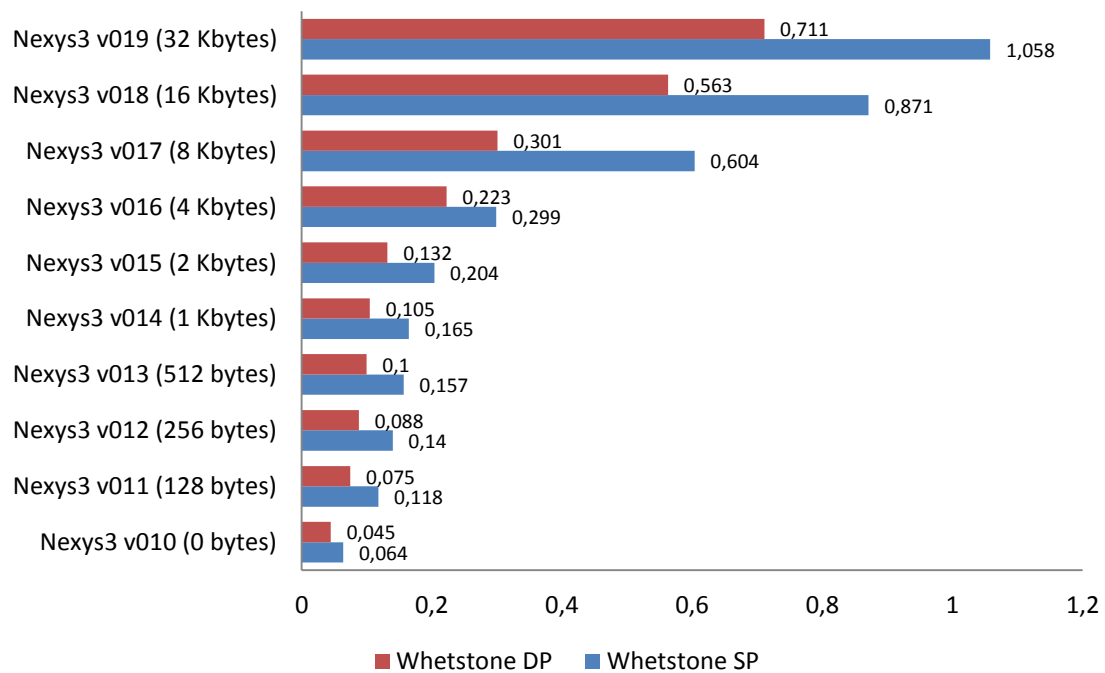


Ilustración 34: Benchmark Whetstone para comparativa de tamaños de caché en Nexys3

Con la implementación de los diversos tamaños de caché, se ha conseguido incrementar el rendimiento de la arquitectura de una manera espectacular. El rendimiento se ha multiplicado por 13 para Dhrystone 2.1, por 8 para Linpack en precisión simple y por 16 de media para Whetstone en precisión simple y doble.

Cabe destacar que estos resultados son cuanto menos importantes, puesto que así se demuestra que el uso de memorias caché incrementa el rendimiento de los soft-processors en gran medida. Esta mejora se produce linealmente o exponencialmente según el benchmark que se utilice.

Analizando exhaustivamente el resultado obtenido para Linpack, se puede ver que ocurren dos sucesos extraños:

- **Linealidad interrumpida:** la línea de progresión que tiende a seguir la mejora en rendimiento para el benchmark se ve truncada cuando en la caché se tienen 8 y 16 KBytes. Para intentar comprender qué ocurre, analizaremos en profundidad otros aspectos como la ocupación de la FPGA, o los tiempos calculados internos de la FPGA.
- **Código sin desenrollar más eficiente que el desenrollado:** el rendimiento obtenido ha sido mejor con el código sin el desenrollado manual que con el código desenrollado manualmente, en las implementaciones hasta 2Kbytes de caché de instrucciones y datos. Este suceso es plausible, puesto que puede ocurrir que el código desenrollado ocupe más espacio que el no desenrollado, lo que implicaría más fallos en la caché de instrucciones, empeorando el rendimiento.

La ocupación de la FPGA ha evolucionado de la siguiente forma:

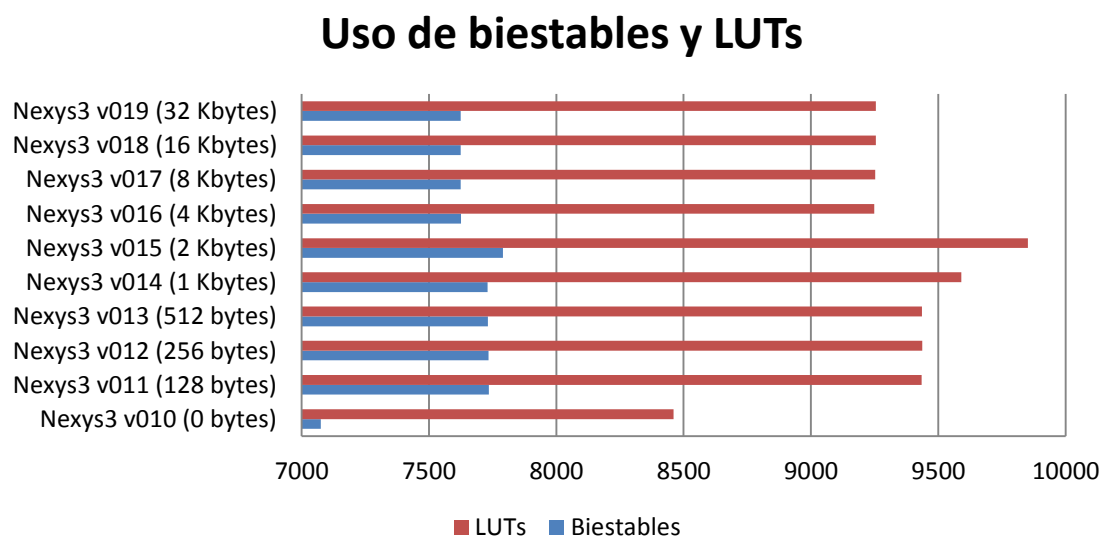


Ilustración 35: Ocupación de FPGA en Nexys3 con distintos tamaños de caché

Uso de BRAM

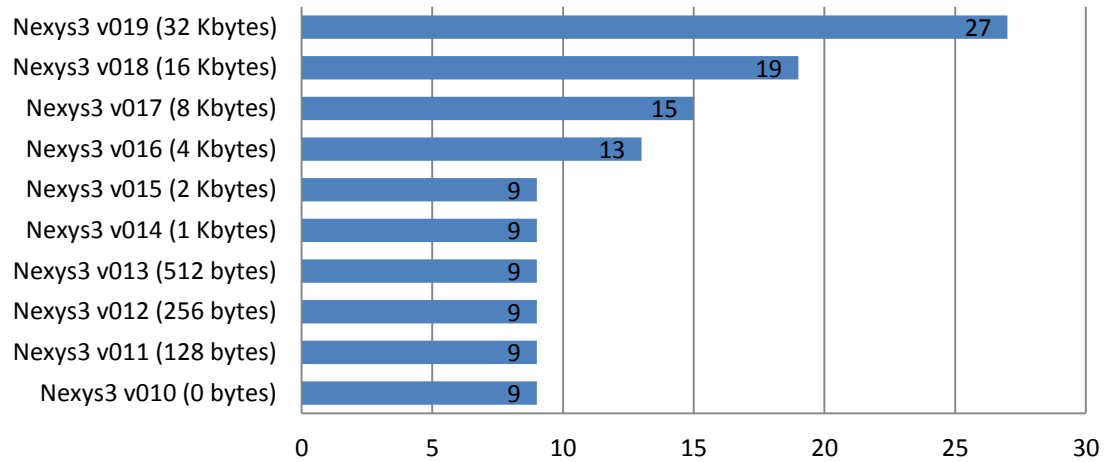


Ilustración 36: Ocupación de bloques BRAM en Nexys3 para distintos tamaños de caché

Como se puede contemplar, los usos de biestables y LUTs no crecen siguiendo la tendencia según se incrementa los tamaños utilizados en la caché, aunque sí que se puede apreciar que el número de biestables y LUTs se incrementan cuando no se utilizan bloques adicionales de BRAM para su uso en las cachés. Esto indica que hasta 2 KBytes de caché estas quedan implementadas en RAM distribuida, y que a partir de dicha cantidad esta queda implementada en módulos de BRAM. También se puede apreciar que la implementación de los módulos de caché tanto si es distribuida como de bloque incrementa el uso de biestables y de LUTs. Este incremento se ve más que recompensado con el rendimiento que ofrece la implementación del módulo.

La ocupación de la FPGA, por tanto, no parece ser el problema que ocasione que los resultados de Linpack con 16Kb y 8Kb de caché sean peores. Analizando los tiempos internos de la FPGA y los problemas de enrutado de los componentes, no se ha llegado a descubrir el porqué de este suceso.

Para finalizar la primera iteración, se va a comprobar la escalabilidad de la versión de Nexys3 que ha obtenido mejor resultado. Utilizaremos como base la versión 19, disminuyendo e incrementando las frecuencias del procesador MicroBlaze:

Versión de FPGA	Frecuencia
Nexys3 v024	50 MHz
Nexys3 v025	62.5 MHz
Nexys3 v019	66 MHz
Nexys3 v026	75 MHz
Nexys3 v027	83 MHz

Tabla 33: Frecuencias de procesador de Nexys3 v019, v024, v025, v026 y v027

Los resultados obtenidos para la escalabilidad han sido los siguientes para los benchmarks Dhrystone 2.1 y Linpack:

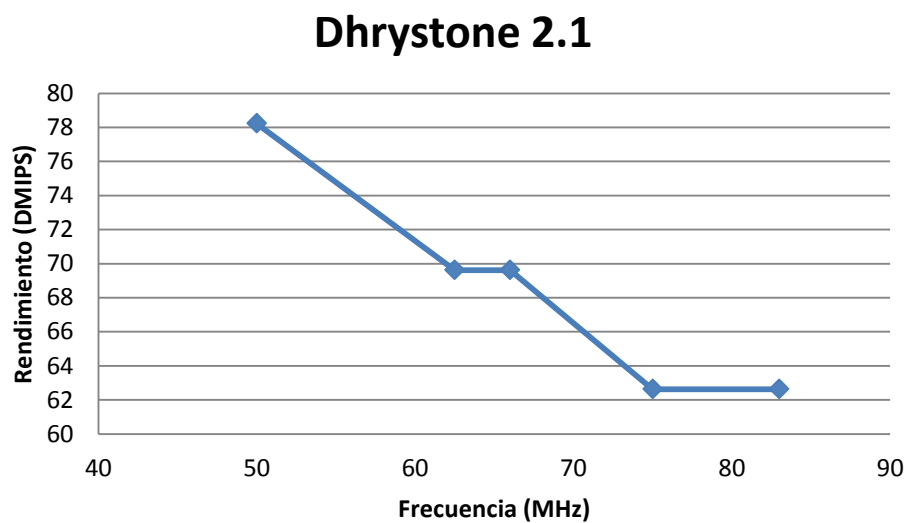


Ilustración 37: Escalabilidad de Nexys3 en términos de frecuencia (Dhrystone)

Linpack

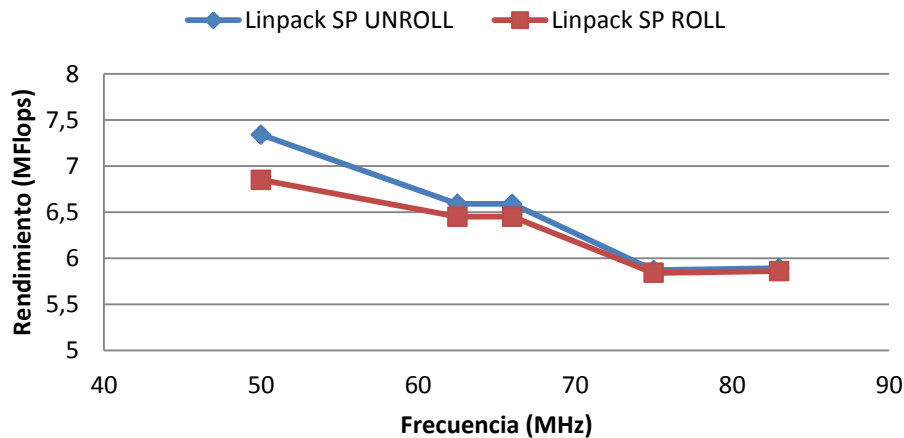


Ilustración 38: Escalabilidad de Nexys3 en términos de frecuencia (Linpack)

Como es posible observar, la escalabilidad es inversamente proporcional, disminuyendo el rendimiento al aumentar la frecuencia del procesador, suceso cuanto menos difícil de comprender. Este descenso del rendimiento llega a situarse en un máximo del 24% para ambos benchmarks, y también para el benchmark Whetstone en sus dos variantes.

Se presentan dos hipótesis que pueden explicar este suceso:

1. **Baja frecuencia de la memoria RAM:** La memoria utilizada por la Nexys3 es del tipo Cellular RAM, un tipo de PSRAM que puede trabajar tanto en modo asíncrono (70ns lecturas y escrituras) como síncrono (máxima velocidad de bus de 80 MHz)^[31]. La opción por defecto establecida por Digilent es el uso de la memoria en modo asíncrono, utilizando la **misma frecuencia de bus que el procesador** para el controlador. Esto puede ocasionar una descompensación entre la velocidad del bus, y la velocidad real a la que funciona la memoria, unos 14MHz internos, causando cuellos de botella que impedirían aprovechar al 100% el rendimiento del procesador. Además el controlador es propietario de Digilent, por lo que no está asegurado el óptimo rendimiento del mismo.
2. **Implementación del diseño hardware en la FPGA poco eficiente:** El proceso de mapeado del diseño hardware puede ser muy ineficiente cuando se están llegando a los límites de implementación en la FPGA. En este caso, el total de memoria BRAM usada es prácticamente el máximo que soporta la Spartan6 LX16, por lo que el mapeo del diseño muy probablemente no es eficiente, y no es capaz de obtener un buen rendimiento a pesar de la configuración establecida.

Para finalizar la primera iteración, vamos a tomar como base los diseños **Nexys3 v024 y v019**, dado que han sido los mejores diseños para la siguiente iteración.

6.2 Segunda iteración

En esta iteración parte de los resultados y conocimientos obtenidos de la primera iteración, aplicándolos a la placa de desarrollo Digilent Atlys. Además, evaluaremos el rendimiento obtenido en los benchmarks VitiRAM y VitiJumps, con el objetivo de obtener mejores conclusiones sobre la arquitectura.

Primeramente se va implementar el mismo diseño implantado en los modelos Nexys3 v019 y v024, a una frecuencia base de 66 MHz, con otros dos diseños incrementando el **tamaño de la caché hasta los 128 Kb** (64 KB de datos y 64 KB de instrucciones) siendo este el máximo tamaño soportado por el procesador MicroBlaze. El motivo de no implementar estos tamaños de caché en la Nexys3 es la poca BRAM disponible en dicha placa (576 KBits), la cual deja un máximo de 63 KB para su uso en cachés de datos e instrucciones.

Las configuraciones son las siguientes:

Versión de FPGA	Tamaño de caché
Nexys3 v019 y v024	16 KB caché de instrucciones 16 KB caché de datos
Atlys v001	16 KB caché de instrucciones 16 KB caché de datos
Atlys v002	32 KB caché de instrucciones 32 KB caché de datos
Atlys v003	64 Kb caché de instrucciones 64 KB caché de datos

Tabla 34: Tamaños de caché para Nexys3 v019 y v024, y Atlys v001, v002 y v003

Y los resultados obtenidos por los benchmarks han sido los siguientes:

Dhrystone 2.1

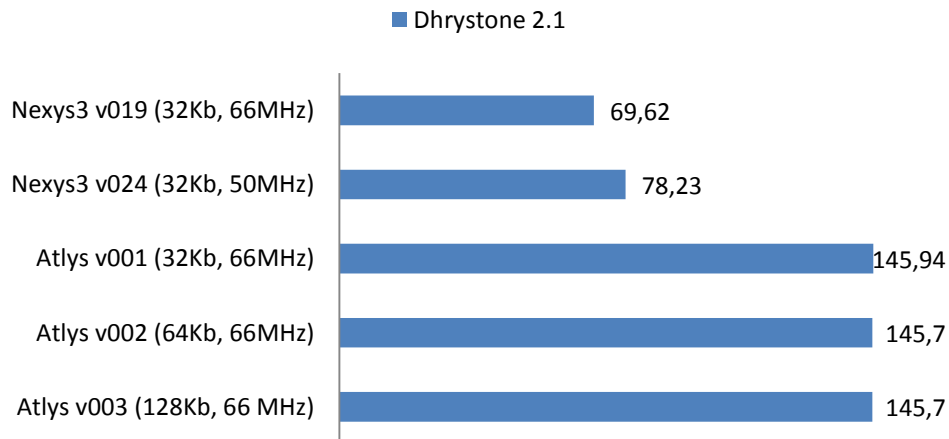


Ilustración 39: Resultado de Dhrystone 2.1 para Nexys3 v019 y v024, y Atlys v001, v002 y v003

Linpack

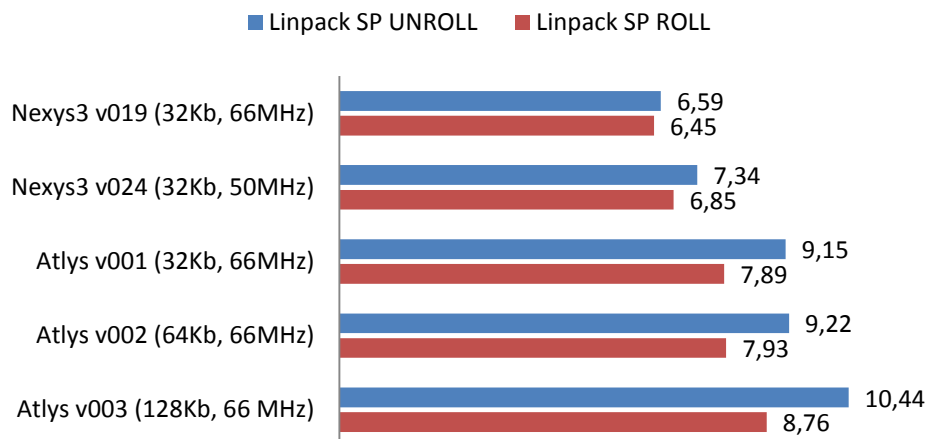


Ilustración 40: Resultado de Linpack para Nexys3 v019 y v024, y Atlys v001, v002 y v003

Whetstone

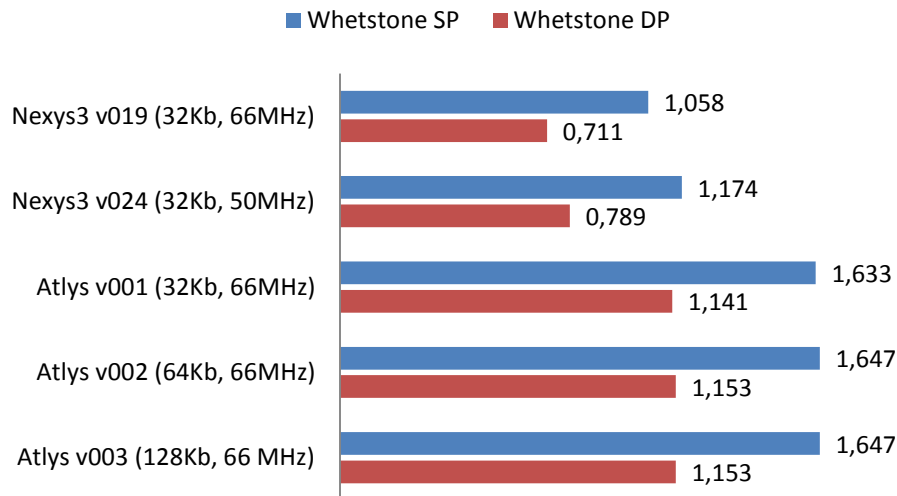


Ilustración 41: Resultado de Whetstone para Nexys3 v019 y v024, y Atlys v001, v002 y v003

De estos resultados es realmente interesante que la misma implementación del procesador MicroBlaze en las placas Nexys3 y Atlys consigue resultados distintos para todos los benchmarks (Nexys3 v019 y v024 respecto la Atlys v001). La diferencia de rendimiento entre ambas placas (Nexys3 v024 y Atlys v001) tiene un valor máximo del 86% en el benchmark Dhrystone 2.1, un 24% para Linpack y un 39% para Whetstone.

Estas diferencias de resultados se deben en gran medida al distinto **tipo de memoria** que utilizan ambas FPGAs, concretamente la Nexys3 utiliza memoria de tipo PSRAM a baja frecuencia, mientras que la Atlys utiliza memoria de tipo DDR2 a alta frecuencia. Se ha de destacar además de que la Atlys utiliza el controlador de memoria integrado en la FPGA Spartan 6, mientras que la Nexys3 utiliza un controlador propio de Digilent. Otro aspecto que influye es la **diferencia de tamaño** de las FPGAs que llevan, mientras que la Spartan 6 LX16 es de un tamaño reducido, la Spartan 6 LX45 tiene un mayor tamaño, lo que la permite distribuir de mejor manera en su interior los componentes del MicroBlaze.

En cuanto al incremento de los tamaños de memoria caché, el rendimiento no mejora sensiblemente, con la excepción del benchmark Linpack que obtiene un resultado mejor en un 15%. Esto se puede deber al pequeño tamaño de los benchmarks, podemos suponer que en aplicaciones de uso común de mayor tamaño el rendimiento sí mejoraría. Sin embargo, consideraremos que a partir de 32 KBytes de caché de instrucciones más datos es una buena configuración.

Ahora se va a analizar el rendimiento en términos de memoria, para asegurarnos de lo comentado anteriormente, comparando las versiones Nexys3 v024 y Atlys v001:

VitiRAM - Escritura en memoria

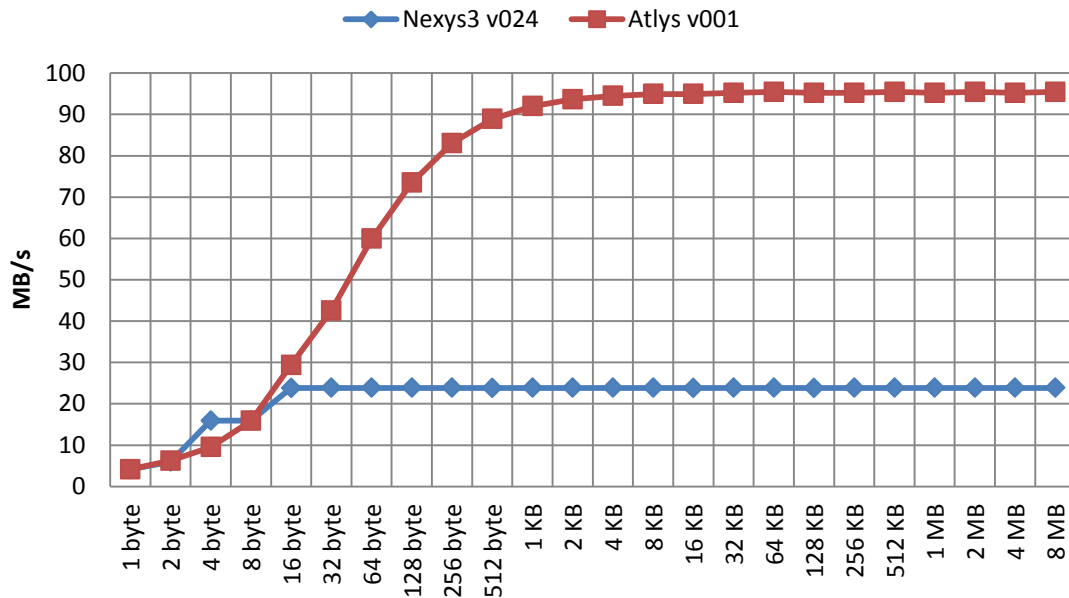


Ilustración 42: Resultado de VitiRAM en escritura para Nexys3 v024 y Atlys v001

VitiRAM - Copia de datos

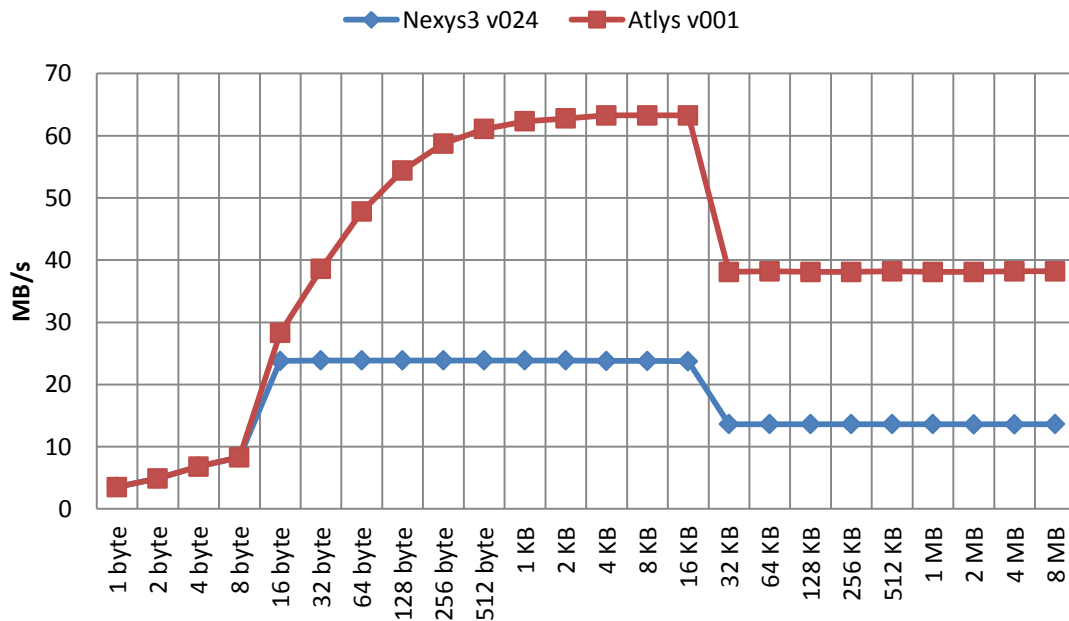


Ilustración 43: Resultado de VitiRAM en copia para Nexys3 v024 y Atlys v001

Como se puede observar, es clara la problemática en términos de transferencia de memoria de la Digilent Nexys3 respecto de la Digilent Atlys. La transferencia de datos es más del doble de eficiente en la Digilent Atlys, con lo que se puede determinar que el **tipo y frecuencia de la memoria RAM** es el factor clave para determinar la diferencia de rendimiento entre ambas placas.

Otro aspecto que se puede observar de la primera de las gráficas anteriores (escritura) es que el tipo de caché de datos es del tipo write-through, ya que el uso de la caché de datos no queda reflejado en dicha gráfica, mientras que en la segunda gráfica sí (copia de datos). Más adelante comprobaremos qué rendimiento se obtiene si se cambia el tipo de caché de datos de write-through a write-back, la cual no realiza una escritura en memoria RAM hasta que no se tenga que reemplazar el bloque en caché.

El siguiente paso es analizar el rendimiento en la predicción de saltos entre la placa Nexys3 v024 y la placa Atlys v001. Los resultados obtenidos determinaron que en ambas placas el resultado es prácticamente el mismo:

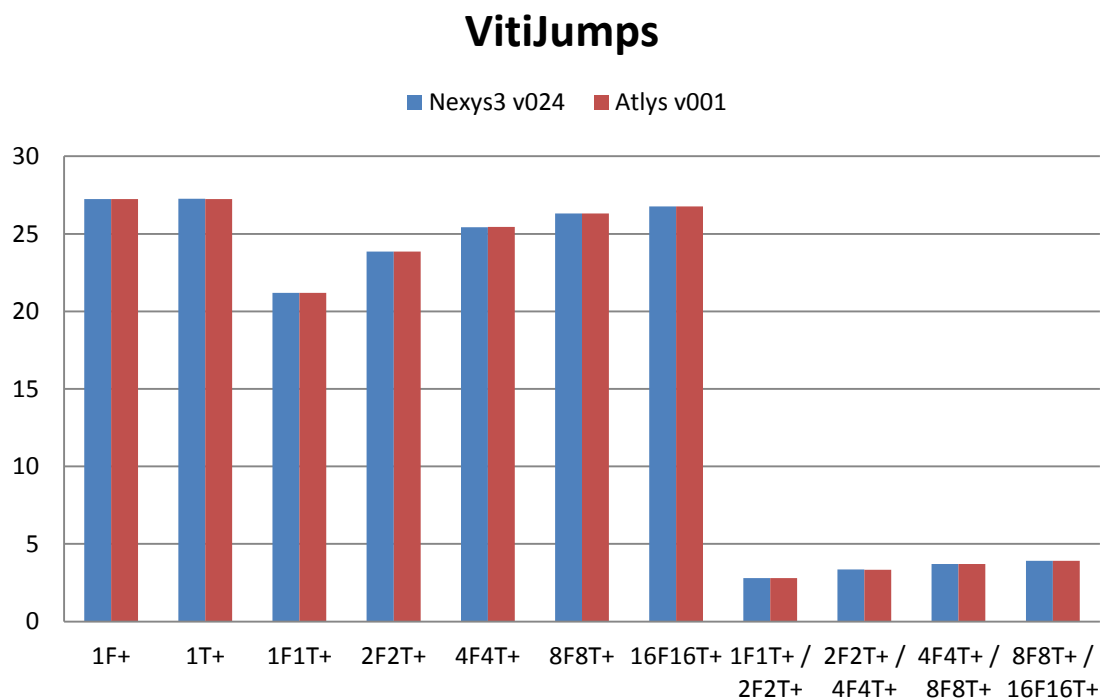


Ilustración 44: Resultado de VitiJumps para Nexys3 v024 y Atlys v001

En el eje de ordenadas de la Ilustración 44 queda reflejado el número de VitiJumps por segundo que el MicroBlaze es capaz de ejecutar, mientras que en el eje de abscisas se representa el configurador de saltos. Los siete primeros indican la eficiencia en saltos directos, mientras que los cuatro últimos los saltos indirectos. La nomenclatura seguida es indicar el número de saltos de un determinado tipo (F para falso, V para verdadero) que se producen consecutivamente en cada iteración.

Un ejemplo de esta nomenclatura sería 8F8V+, lo que indica que se ejecutarían ocho saltos falsos, seguido de ocho verdaderos, y se repetiría esta serie continuamente.

En la gráfica se puede apreciar como la predicción de saltos es independiente de la placa en la que se implemente, y además es bastante buena para este procesador, puesto que la predicción de saltos continuados mejora según se alargan las repeticiones de las condiciones en los bucles de control *if-then-else*. Queda también claro que la peor situación que se le puede plantear a un procesador MicroBlaze es la alternancia de condiciones verdaderas y falsas en bucles *if-then-else*.

Ahora vamos a analizar el rendimiento que se alcanza si se cambia el tipo de caché de datos de write-through a write-back. Para ello se va a utilizar el diseño de la Atlys v002 y se va a modificar de la siguiente manera:

Versión de FPGA	Tipo de caché
Atlys v002	Write-through
Atlys v007	Write-back

Tabla 35: Políticas de caché en Atlys v002 y v007

Los resultados obtenidos para los benchmarks de ALU y FPU fueron los siguientes:

Prueba de rendimiento	Resultado de Atlys v002	Resultado de Atlys v007
Dhrystone 2.1	145.7 DMIPS	173.13 DMIPS
Linpack SP Unroll	9.22 MFlops	9.62 MFlops
Linpack SP Roll	7.93 MFlops	8.17 MFlops
Linpack DP Unroll	0.25 MFlops	0.29 MFlops
Linpack DP Roll	0.25 MFlops	0.29 MFlops
Whetstone SP	1.647 MWIPS	1.841 MFlops
Whetstone DP	1.153 MWIPS	1.299 MFlops

Tabla 36: Rendimiento en Atlys v002 y v007

Y la diferencia de rendimiento relativo de una respecto de la otra es la siguiente:

Rendimiento relativo entre Atlys v002 y v007

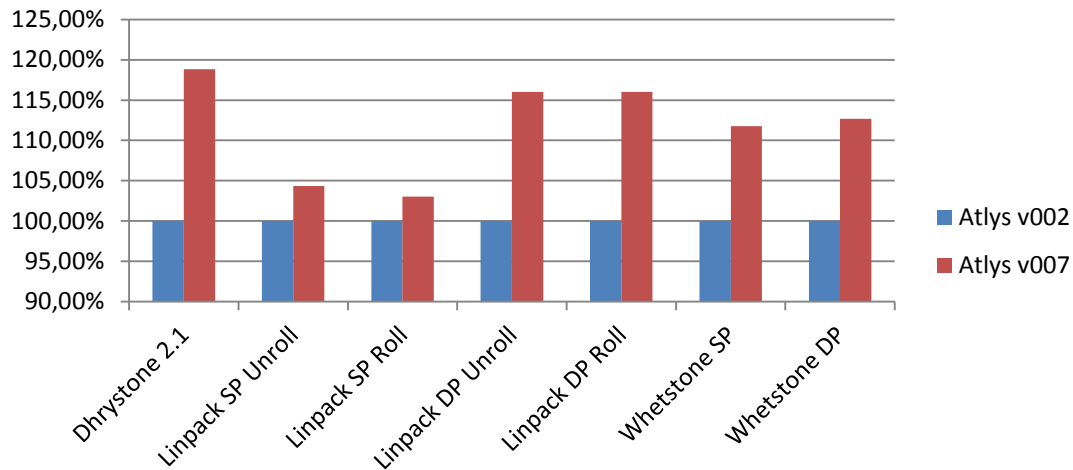


Ilustración 45: Rendimiento relativo entre Atlys v002 y Atlys v007

A simple vista, se puede observar que el uso de la caché de tipo write-back mejora los resultados de los benchmarks respecto de la caché write-through en un margen del 3% al 18%. Ahora vamos a comparar el rendimiento en términos de transferencias de memoria:

VitiRAM - Escritura

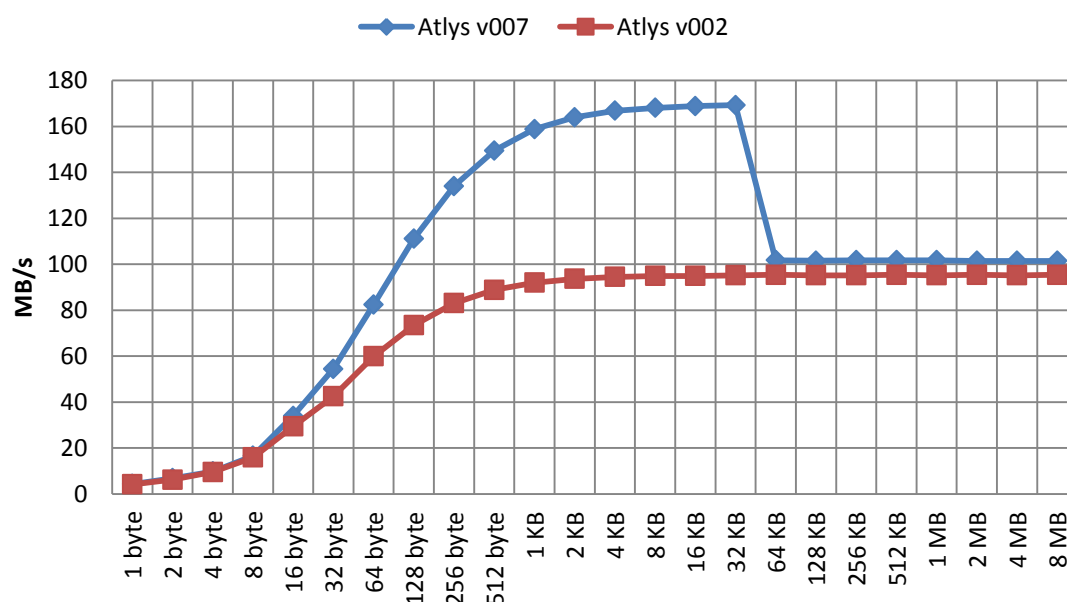


Ilustración 46: Resultado de VitiRAM en escritura para Atlys v002 y Atlys v007

VitiRAM - Copia de datos

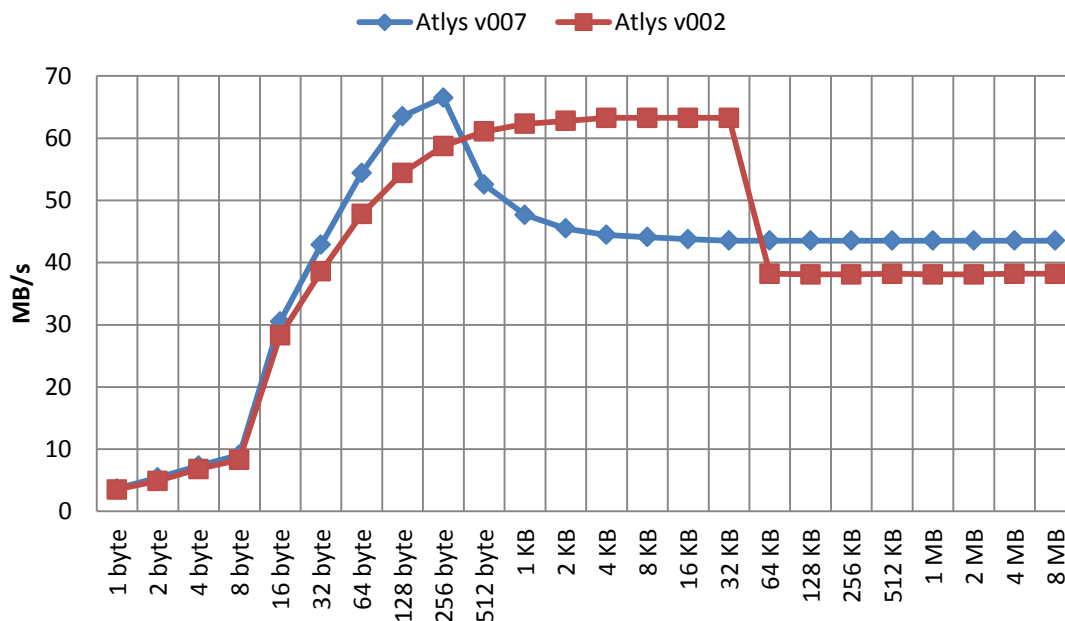


Ilustración 47: Resultado de VitiRAM en copia para Atlys v002 y Atlys v007

Como se puede observar, la implementación de una caché de tipo write-back ha ocasionado la obtención de distintos resultados para las transferencias de memoria. En el caso de las escrituras, ésta se ha mejorado considerablemente, puesto que las escrituras sólo se realizan en memoria RAM cuando la caché ha quedado completamente llena. Sin embargo, los procesos de copia de datos se han empeorado en cierta medida, sobre todo a partir de la de los 256 bytes hasta el tamaño máximo de la caché.

Este suceso ocurre debido a que en todo momento se están produciendo fallos de caché a la hora de leer de memoria, lo que lleva a que primero se realice una escritura de caché a memoria (en el caso de que el bloque de datos halla sido modificado), y posteriormente se lea el bloque desde memoria. Esto implica una pérdida de rendimiento considerable. Esta situación se puede solucionar con la implementación de una caché de tipo *victim*, que buscará almacenar los últimos datos reemplazados en el caso de que sea necesario volver a ser utilizados.

El siguiente paso a realizar es comprobar la efectividad del predictor de saltos, utilizando diversas configuraciones del mismo (número de saltos que puede predecir y su implementación en BRAM o RAM distribuida) y con el predictor desactivado, con la idea de mejorar este modulo.

Las configuraciones se basan al igual que en el caso anterior en la Digilent Atlys v002, de la siguiente manera:

Versión de FPGA	Predictor de saltos
Atlys v008	Desactivado
Atlys v010	BTC – 64 entradas – RAM Distribuida
Atlys v002	BTC – 256 entradas – BRAM
Atlys v009	BTC – 2048 entradas – BRAM

Tabla 37: Configuración del predictor de saltos en Atlys v002, v008, v009 y v010

Los resultados para el benchmark VitiJumps han sido los siguientes:

VitiJumps - Predictor BTC

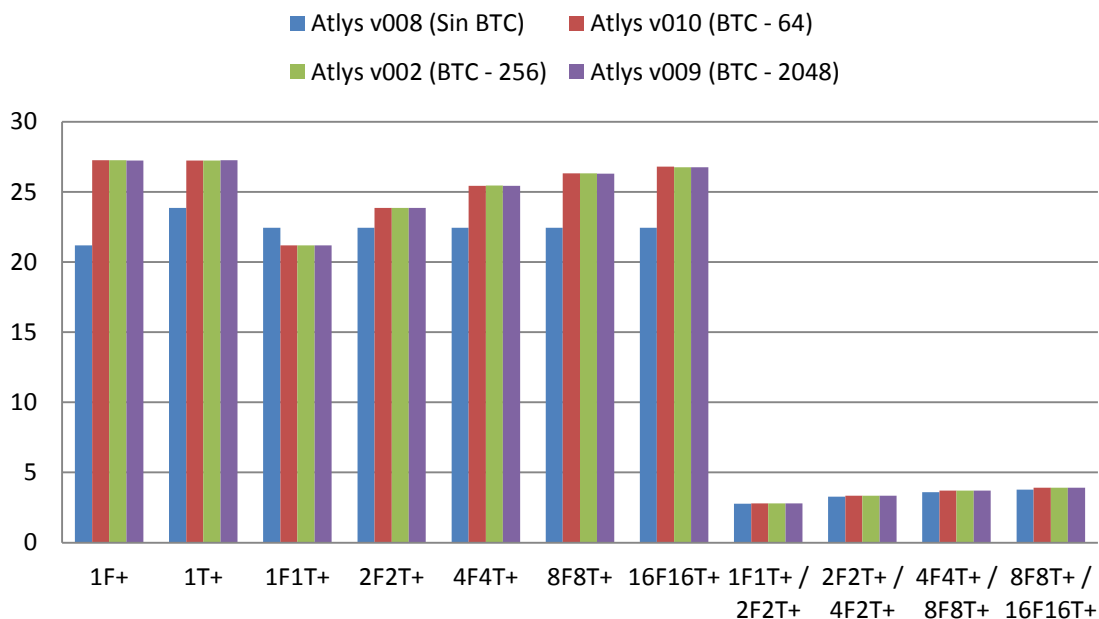


Ilustración 48: Resultado de VitiJumps para Atlys v002, v008, v009 y v010

Se puede apreciar como la implementación del predictor de saltos mejora el rendimiento en el benchmark para los saltos directos, con la excepción de los saltos intercalados verdadero y falso (1F1T+) y para los saltos indirectos. Esto ocurre dado que una predicción de saltos errónea en el BTC implica una penalización de dos ciclos de reloj, mientras que sin tener un BTC implementado esto no ocurre.

Por otro lado, los distintos tamaños del BTC no afectan de manera significativa el rendimiento para el benchmark VitiJumps, aunque sí que podrían afectar a programas de un mayor tamaño.

Para finalizar esta iteración, se implementa una versión de Digilent Atlys con la mejor configuración posible vistos los resultados. Los datos técnicos finales de esta versión son los siguientes:

- **Frecuencia de reloj:** 66 MHz
- **Caché de instrucciones y datos:** 64 KB para instrucciones y 64 KB para datos (128 KB en total), longitud de línea de caché de 8 palabras, buffer de instrucciones adicional de tipo *stream* y búferes de tipo *victim* para instrucciones y para datos. Política de reemplazo *write-back* para la caché de datos.
- **Unidad de multiplicación de enteros de 64 bits**
- **Unidad de división de enteros de 32 bits**
- **Unidad de coma flotante de precisión simple extendida**
- **Instrucciones adicionales:** comparación de bits, desplazamiento de bits y control avanzado para el MSR
- **Buffer de predicción de saltos:** 2048 entradas

La denominaremos **Atlys v017**, y los resultados finales que se obtuvieron para dicha placa, comparadas con las mejores implementaciones realizadas, han sido los siguientes:

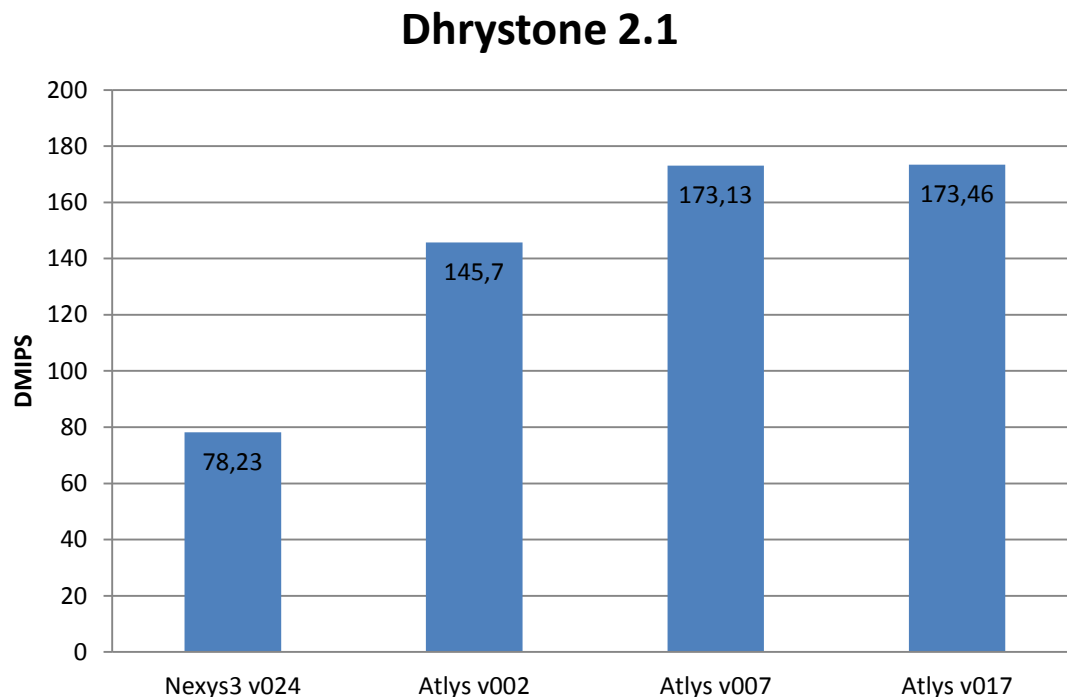


Ilustración 49: Resultado de Dhrystone 2.1 para Nexys3 v024 y Atlys v002, v007 y v017

Linpack

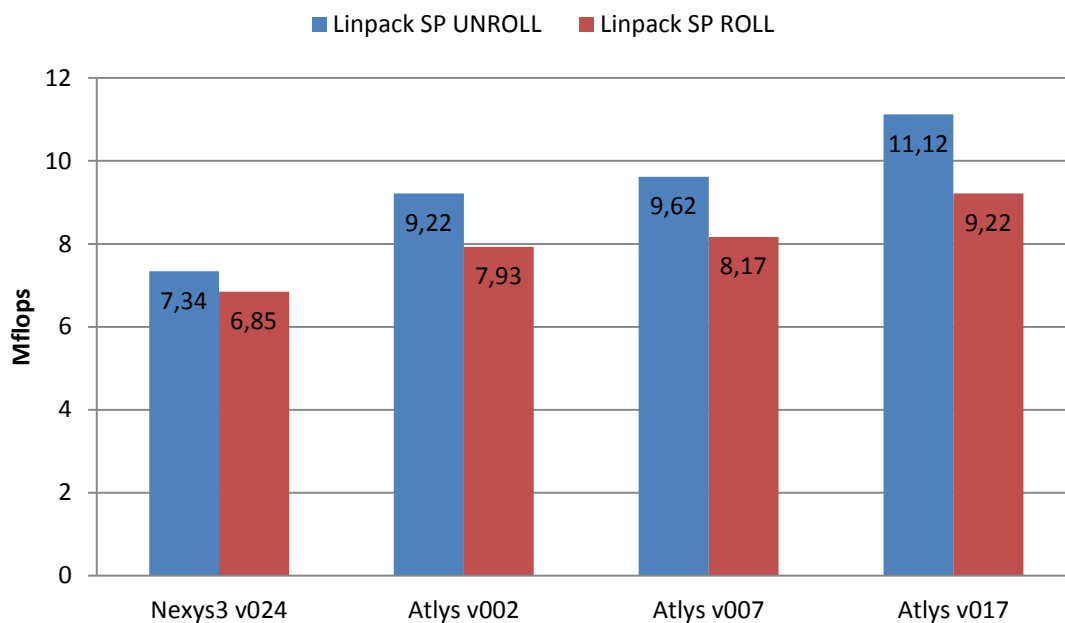


Ilustración 50: Resultado de Linpack para Nexys3 v024 y Atlys v002, v007 y v017

Whetstone

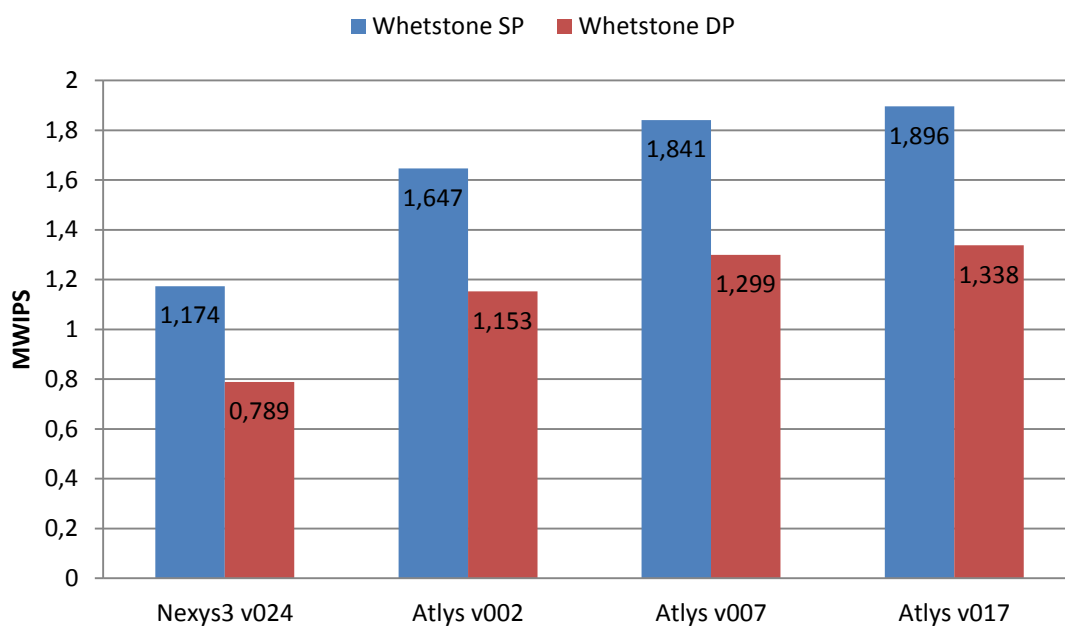


Ilustración 51: Resultado de Whetstone para Nexys3 v024 y Atlys v002, v007 y v017

A simple vista queda claro que la implementación de dichas mejoras no ha supuesto una gran mejora respecto a la versión v007, consiguiéndose la mayor mejora en el benchmark Linpack, con un 15% para la versión desarrollada en coma flotante simple, y un 13% para la versión sin optimizar. El resto de benchmarks, Dhrystone 2.1 y Whetstone se han visto mínimamente mejorados. En cuanto al ancho de banda de transferencias de datos, estos fueron los resultados:

VitiRAM - Escritura

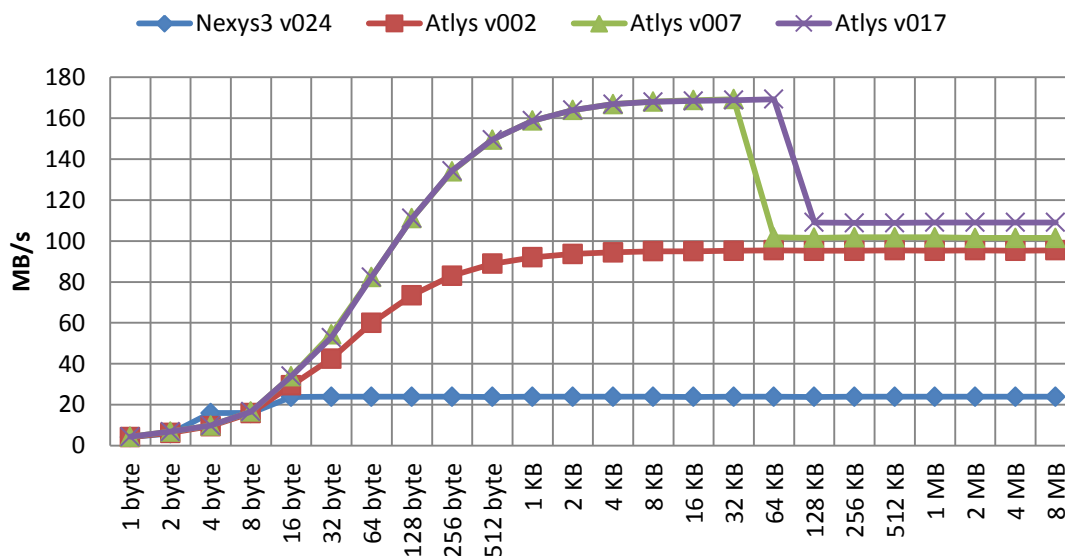


Ilustración 52: Resultado de VitiRAM en escritura para Nexys3 v024, y Atlys v002, v007 y Atlys v017

VitiRAM - Copia de datos

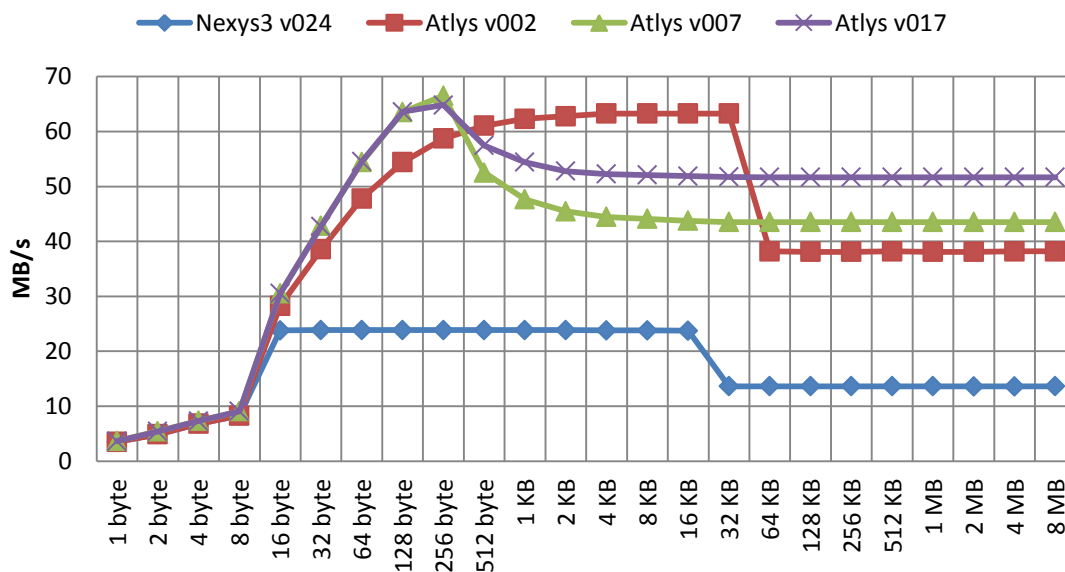


Ilustración 53: Resultado de VitiRAM en copia para Nexys3 v024, y Atlys v002, v007 y Atlys v017

Las transferencias de datos se han visto mejoradas respecto de la versión v007, al incrementarse el tamaño de caché de datos de 32Kb a 64Kb en el caso de las escrituras. También se han visto mejoradas en la copia de datos, aunque el rendimiento es un poco inferior al obtenido con la política de reemplazo write-through para las transferencias de datos en las que el tamaño a transferir esté entre los 512 bytes y el máximo de la caché de datos.

Finalmente, se considera la versión **v017** de Atlys como la mejor de todas, al obtener el mejor rendimiento posible de todas las creadas. Se comprobó que incrementando la frecuencia del procesador a 75MHz o a 83MHz en esta versión no supuso mejora alguna (no como sucedía en la Nexys3), por lo que esta versión será la que utilizaremos para la tercera iteración.

6.3 Tercera iteración

El último paso para conseguir la mejor optimización del soft-processor MicroBlaze, es realizar diversas **optimizaciones en el compilador** que utiliza el Xilinx SDK sobre la plataforma Atlys v017. El compilador es una versión adaptada de GCC, por lo que se pueden utilizar las siguientes optimizaciones^[50]:

- **-Ox**: aplica optimizaciones diversas al proceso de compilación. Estas optimizaciones aumentan el tiempo de compilación y el espacio ocupado en memoria RAM al compilar. El valor *x* puede tener los siguientes valores:
 - **-O0**: ninguna optimización.
 - **-O1**: optimización simple.
 - **-O2**: optimización más avanzada que -O1, realiza casi todas las optimizaciones que no impliquen un compromiso de espacio respecto de la velocidad de lo compilado.
 - **-O3**: optimización mayor, activa todas las optimizaciones de -O2 y activa otras optimizaciones que no consideran el tamaño del código.
 - **-Os**: optimización para el espacio, realiza todas las optimizaciones que no incrementan el tamaño del ejecutable compilado respecto a una versión sin optimizar. Incluye además optimizaciones que decrementan el tamaño del compilado.

- **-ffast-math:** esta optimización omite las normas IEEE o ISO para la implementación de las funciones matemáticas, incrementando el rendimiento. Esta optimización se considera peligrosa puesto que no se garantiza que las aplicaciones existentes sigan funcionando.
- **-funroll-loops:** desenrollado automático de bucles con número de iteraciones determinado en tiempo de compilación. Hace que el código sea más grande, y no implica una mejora de rendimiento.
- **-fmodulo-sched:** optimización que reordena las instrucciones en la planificación previa a la generación de código para bucles, con la idea de superponer varias de las iteraciones de los bucles.
- **-ftracer:** realiza una duplicación de las colas para agrandar el tamaño del superbloque (duplica el código). Esta transformación simplifica el flujo de control de las funciones, permitiéndose realizar otras optimizaciones más eficientemente.

Las cuatro últimas optimizaciones consideradas no se aplican automáticamente por las de tipo -Ox, por lo que primeramente vamos a aplicar las distintas optimizaciones -Ox, seleccionar la mejor de ellas y posteriormente comprobar como afectan estas cuatro. Las pruebas devolvieron los siguientes resultados para las pruebas de enteros y coma flotante:

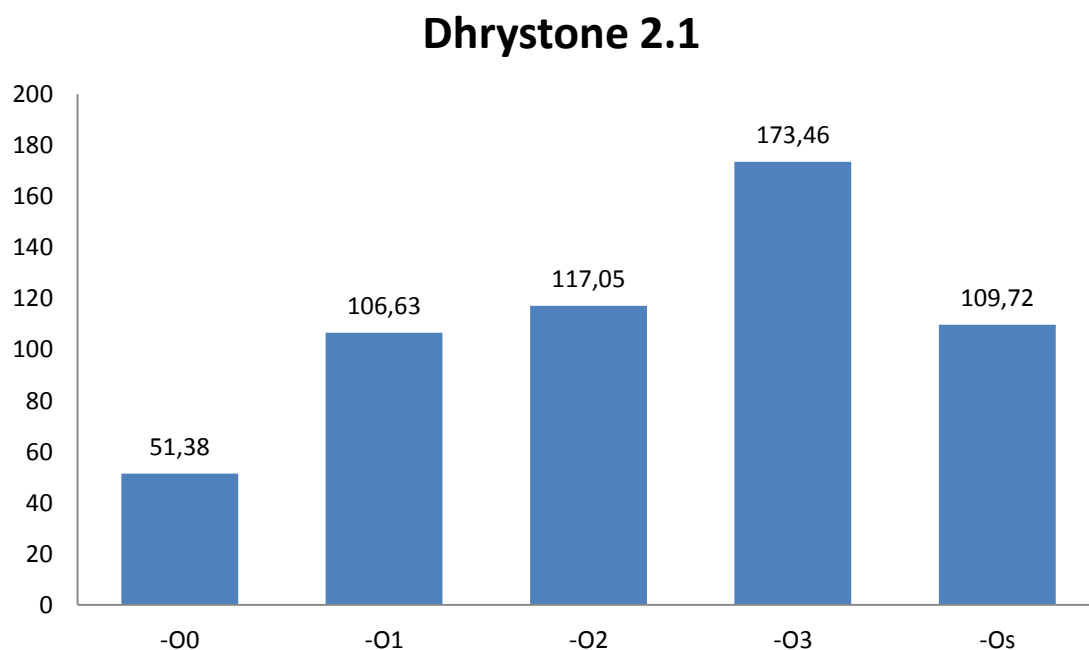


Ilustración 54: Resultado de Dhrystone 2.1 con distintas optimizaciones generales de GCC

Linpack

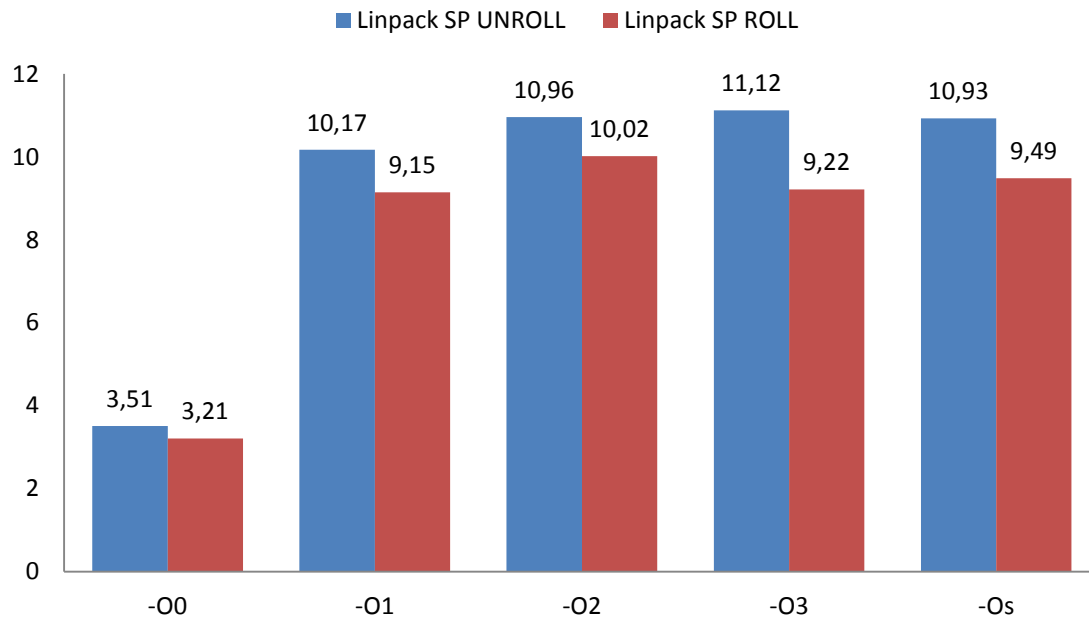


Ilustración 55: Resultado de Linpack con distintas optimizaciones generales de GCC

Whetstone

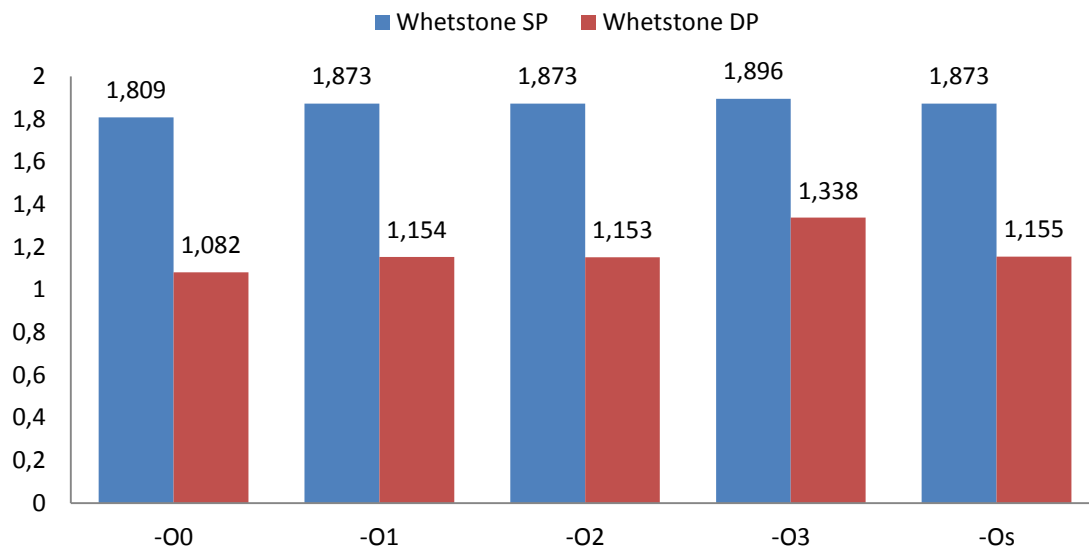


Ilustración 56: Resultado de Whetstone con distintas optimizaciones generales de GCC

A simple vista, se puede observar que la optimización `-O3` es la que mejores resultados obtiene para todos los benchmarks, con la excepción de la versión sin desenrollar de Linpack. Cabe destacar que esta es la opción por defecto que establece el entorno de desarrollo de Xilinx, pero no para el compilador GCC.

Con estos resultados podemos determinar que esta optimización es la que mejor resultado obtiene respecto del resto. Ahora vamos a aplicar las optimizaciones extra incluyendo la optimización -O3, y veremos si se mejoran o empeoran los resultados:

Dhrystone 2.1

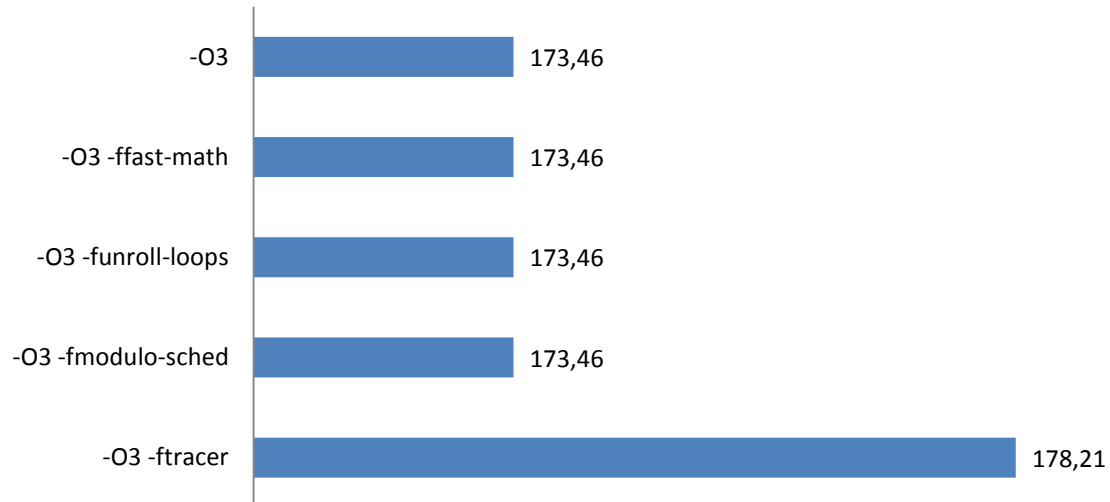


Ilustración 57: Resultado de Dhrystone 2.1 con distintas optimizaciones específicas de GCC

Linpack

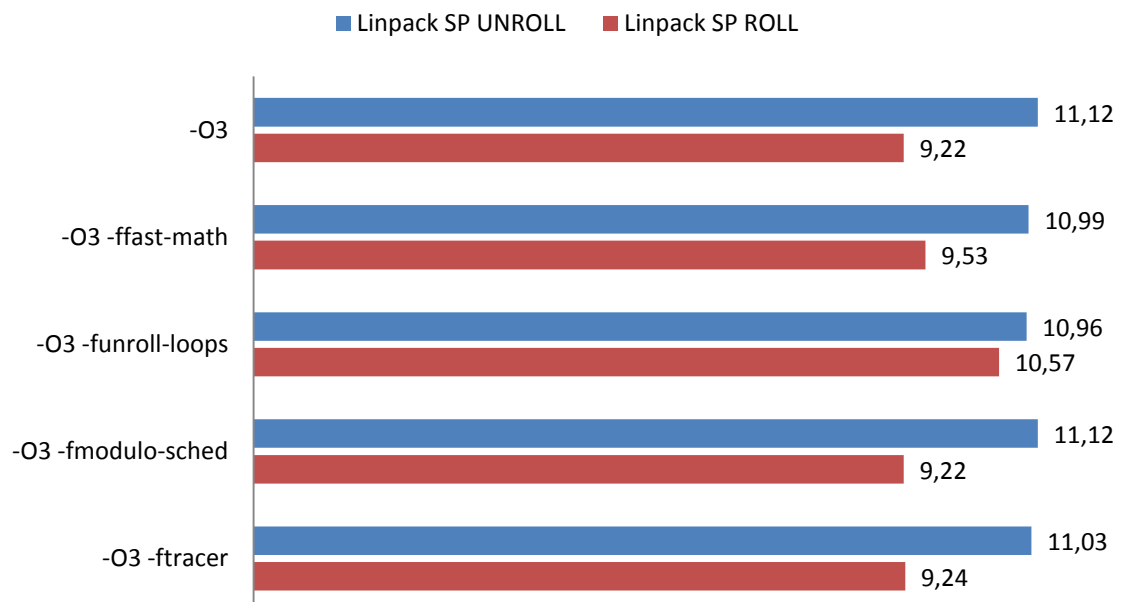


Ilustración 58: Resultado de Linpack con distintas optimizaciones específicas de GCC

Whetstone

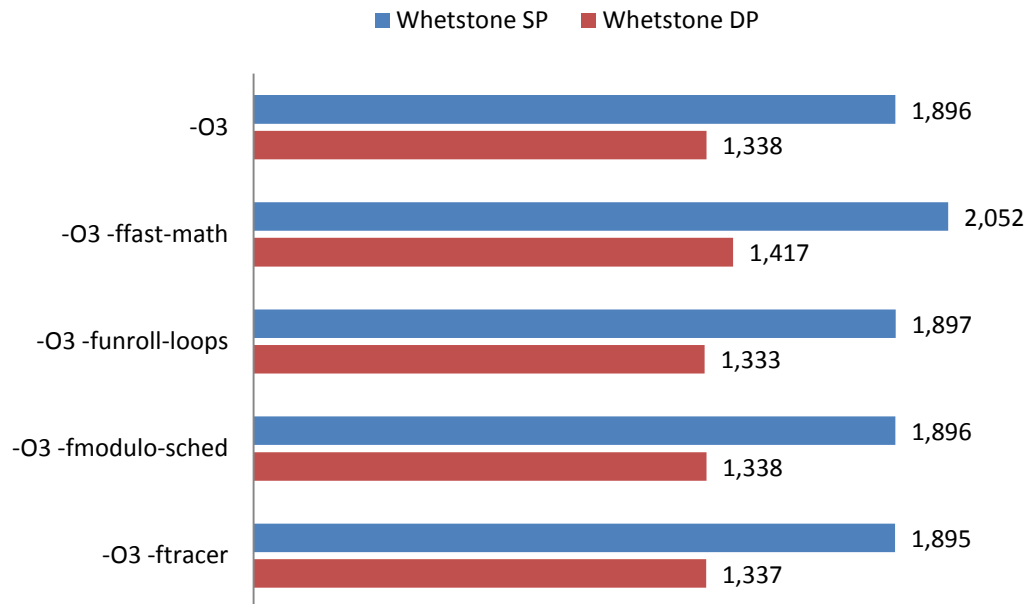


Ilustración 59: Resultado de Whetstone con distintas optimizaciones específicas de GCC

De las pruebas realizadas, cabe destacar que ninguna de las cuatro optimizaciones adicionales utilizadas produce un incremento del rendimiento que se pueda considerar como sustancial para todas las pruebas, sin embargo sí que se pueden observar los siguientes comportamientos para cada una de las opciones:

- La opción **-ffast-math** mejora el rendimiento de Whetstone entre un 5% y un 8%, causado principalmente por la optimización de las funciones trigonométricas. En el resto de benchmarks no ofrece mejora alguna, puesto que no hacen uso de estas funciones, y por otro lado es posible que corrompa la ejecución de los benchmarks.
- La opción **-funroll-loops** mejora el rendimiento de Linpack con código sin desenrollar. Esto es un indicativo de que es capaz de detectar que el código es optimizable con un desenrollado de los bucles, aunque no es capaz de conseguir el mismo rendimiento que en un desenrollado manual. Como vimos en la primera iteración, esta optimización no sería efectiva en el caso de disponer un tamaño reducido de caché, puesto que al implicar tener un código más grande, el número de fallos de caché sería aún mayor.
- La opción **-fmodulo-sched** no conllevó mejora ni detrimento alguno para ninguno de los benchmarks.
- La opción **-ftracer** produce incrementos de rendimiento en las pruebas de enteros, pero disminuye el rendimiento en las que usan coma flotante.

A la vista de los resultados se puede determinar que las optimizaciones adicionales no representan un factor crítico, pero que sí que pueden mejorar el rendimiento ante ciertos programas o justamente todo lo contrario. Es por ello, que la única manera de determinar cuáles son las mejores sería mediante la experimentación, tal y como se está haciendo en este proyecto.

Con estos resultados finalmente podemos pasar a la cuarta y última iteración. La solución elegida por tanto es la placa de Digilent Atlys en su versión v017, con la **optimización –O3 del compilador** para todos los benchmarks.

6.4 Cuarta iteración

En esta última iteración se va a comparar el rendimiento para la placa Digilent Atlys y el soft-processor MicroBlaze implementado, con otras arquitecturas de características similares.

Primeramente se analizará el rendimiento de las unidades de enteros y coma flotante, probando en las arquitecturas con diversas optimizaciones de hardware. Los procesadores probados son de las arquitecturas ARM, MIPS y x86:

- **ARM ARM946E-S:** Procesador usado por la Nintendo DS. Implementa el conjunto de instrucciones ARM5vTE, con soporte de instrucciones de 32 bits ARM y un conjunto de instrucciones de 16 bits Thumb (código más pequeño respecto al conjunto de 32 bits). Esta versión incluye **una unidad DSP**, con instrucciones para la multiplicación-acumulación (resta o suma) y contado de ceros. El procesador probado funciona a una velocidad de 66 MHz, y consta de 4 KBytes de caché de datos, y 8 KBytes de caché instrucciones. Además incluye un espacio propio de caché denominado TCM de 32 KBytes, dividido en dos módulos^{[27][28]}.
- **MIPS Allegrex:** Usado por la Sony Playstation Portable. Incluye el conjunto de instrucciones MIPS32R2 (también conocido como MIPS4k), siendo éste de 32 bits. El procesador incluye 16 KBytes de caché de datos y 16 KBytes de instrucciones, una caché de datos/instrucciones *scratchpad* de 16 KBytes, **una unidad dedicada FPU** para instrucciones de coma flotante, y **una unidad dedicada VFPU** para instrucciones vectoriales. Por otro lado, incluye instrucciones específicas para el procesamiento de gráficos 3D^{[23][24][46]}.

- **Intel 80486DX2 y 80486DX4:** Procesadores de 32 bits que implementan la arquitectura x86, poseen instrucciones adicionales a la normativa x86 (las mismas que el 80386, y dos instrucciones nuevas). Incorpora caché de datos e instrucciones compartida, de 8 KBytes para los primeros modelos (DX y DX2) y 16 KBytes para los posteriores (DX4, los últimos modelos incluyen la política de reemplazo *write-back* para la caché). Además incluye una **FPU de 80 bits de precisión**, con soporte de coma flotante simple y doble^{[13][14][15]}.
- **Intel Pentium P5 y P54C:** Procesador de 32 bits que implementa la arquitectura x86, **bus de datos externo de 64 bits** y con instrucciones adicionales a la normativa x86 (incluye las del procesador 80486DX y varias adicionales, el conjunto es conocido como i586). Es la primera implementación de x86 en la **arquitectura superescalar**, con dos fuentes de datos para poder ejecutar dos instrucciones a la vez. Incluye mejoras en la arquitectura usando **microcódigo**, una unidad de **FPU avanzada de 80 bits**, y cachés de datos e instrucciones separadas de 8 KBytes cada una^{[16][17][18][19]}.

En cuanto a las opciones de compilación utilizadas, éstas han sido las mismas que las establecidas para las arquitectura MicroBlaze, es decir, **-O3**. Dado que esta opción no existe en el entorno de desarrollo OpenWatcom, se ha simulado activando todas las opciones de optimización que permite este compilador^[49].

Ahora vamos a comparar los resultados de estos procesadores con el soft-processor diseñado y mejorado. Primeramente las pruebas de rendimiento de enteros y coma flotante:

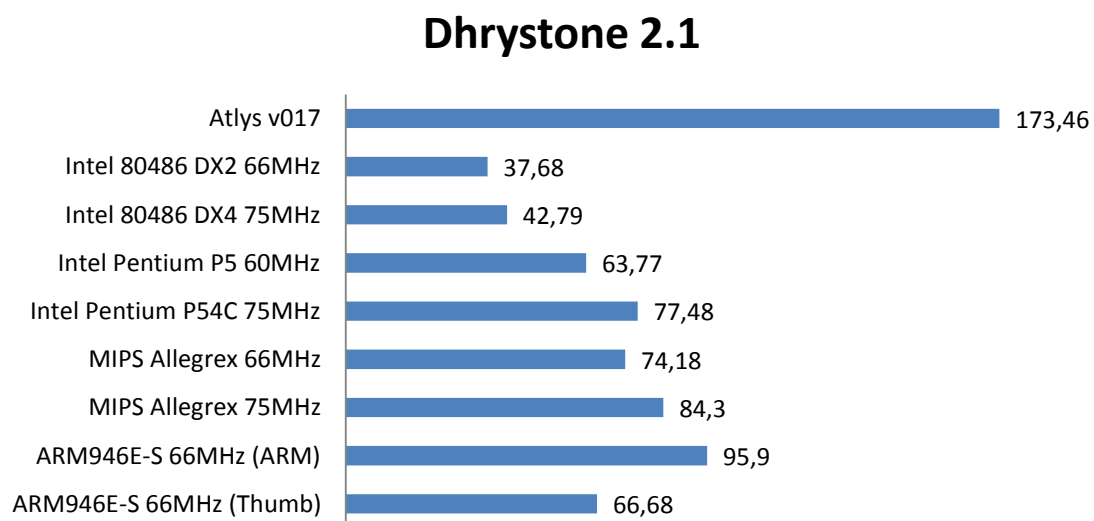


Ilustración 60: Resultados de Dhrystone 2.1 para distintas arquitecturas

Se puede apreciar claramente como el resultado del procesador MicroBlaze es muy superior a todos los demás. Este resultado puede deberse principalmente a la gran caché de datos y de instrucciones que se puede acoplar al MicroBlaze, haciendo que todo el benchmark se ejecute dentro de dicha caché. El peor resultado ha sido el obtenido por los procesadores Intel 80486DX2 y 80486DX4, dado que disponen de una tecnología más anticuada. También se puede observar como las arquitecturas ARM, MIPS y x86 tienen resultados muy parejos, y como el uso del conjunto de instrucciones reducido Thumb del procesador ARM decrementa el rendimiento respecto al conjunto de instrucciones completo.

Linpack

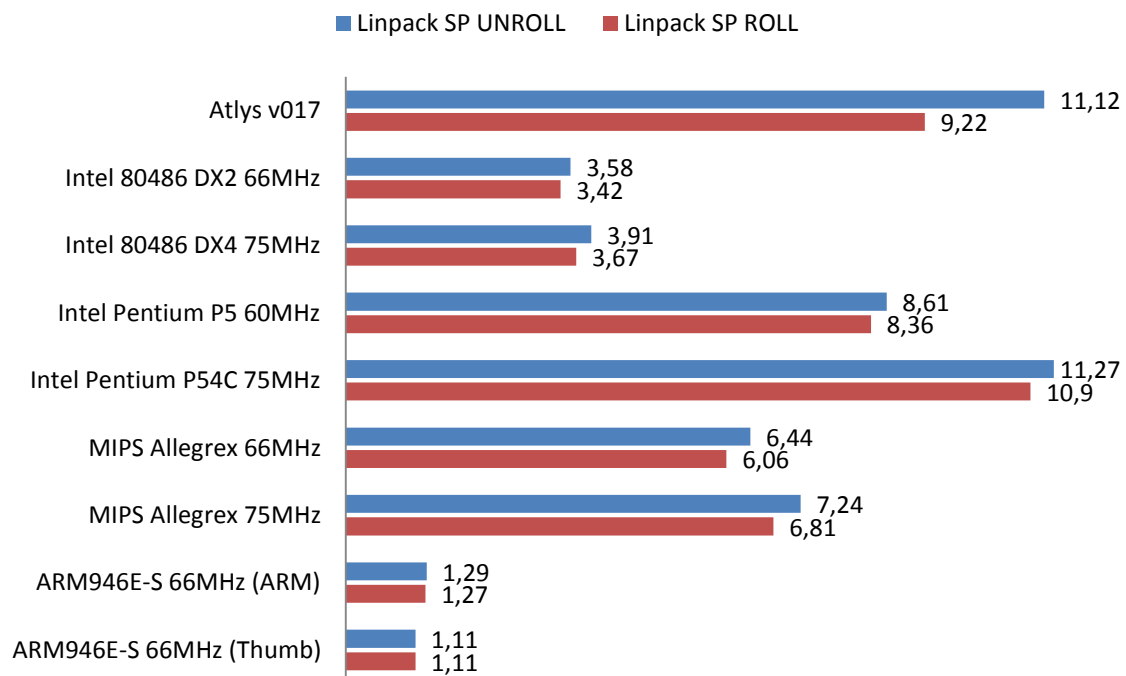


Ilustración 61: Resultados de Linpack en precisión simple para distintas arquitecturas

El rendimiento del procesador MicroBlaze en este caso se encuentra muy a la par por el conseguido por el procesador Intel Pentium de primera generación en coma flotante simple para el benchmark Linpack, superando al resto de arquitecturas. El mal resultado del procesador ARM es totalmente comprensible, puesto que éste carece de una unidad FPU que acelere las operaciones en coma flotante.

Ahora se va a evaluar el resultado del mismo benchmark, en precisión doble:

Linpack

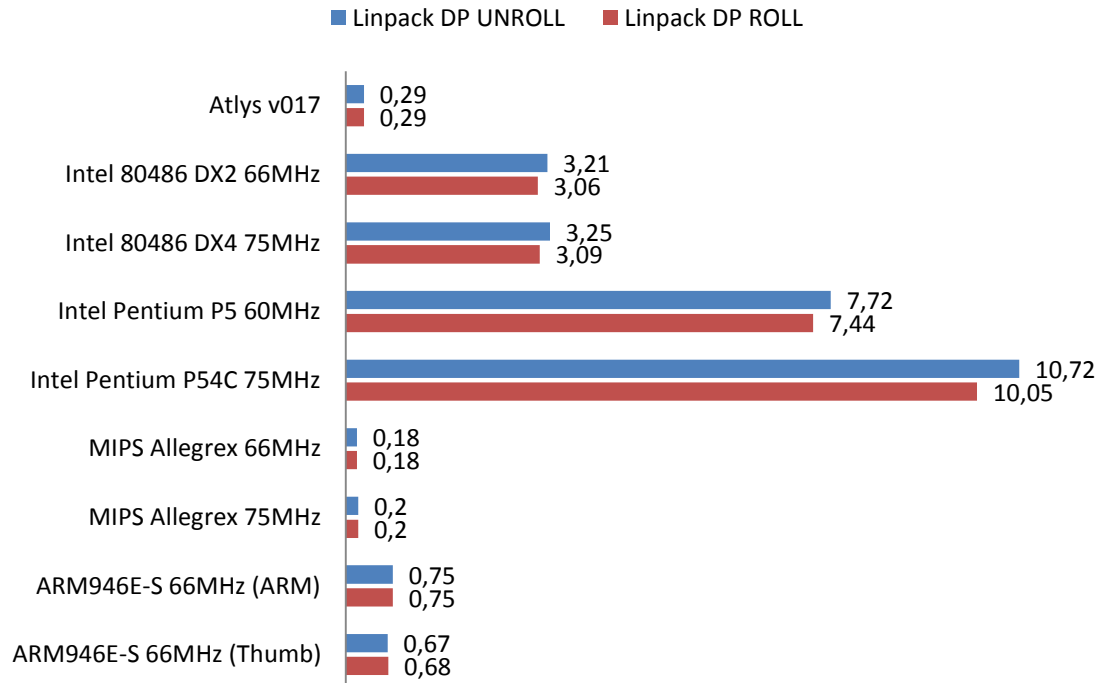


Ilustración 62: Resultados de Linpack de precisión doble para distintas arquitecturas

Comparando los resultados obtenidos en precisión doble respecto los de precisión simple, el rendimiento para coma flotante de doble precisión resulta ser totalmente desastroso, a la par del obtenido por las arquitecturas MIPS y ARM. El motivo principal de estos resultados es que los procesadores utilizados carecen de soporte de instrucciones de coma flotante para precisión doble, mientras que la arquitectura x86 sí que soporta dicha precisión en hardware. Además se puede apreciar cómo la implementación de 80 bits de la FPU de los procesadores Intel no pierde una gran cantidad de rendimiento respecto al mismo benchmark con precisión simple.

El siguiente paso es analizar los resultados obtenidos en el benchmark Whetstone:

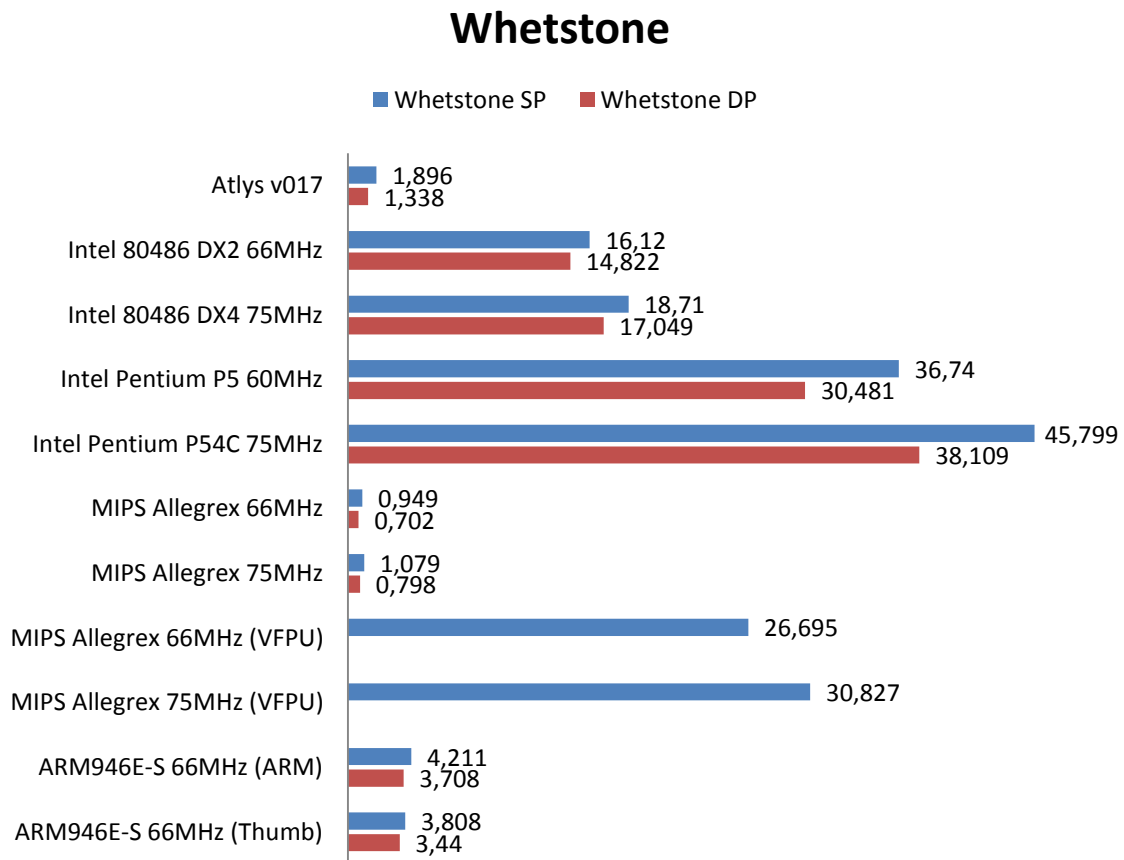


Ilustración 63: Resultados de Whetstone para distintas arquitecturas

Por último, el resultado para el benchmark Whetstone es deficiente respecto al resto de arquitecturas. Este problema se debe en gran medida a la falta de implementación de determinadas instrucciones necesarias para la obtención de un buen resultado en este benchmark. Concretamente, las pruebas demostraron que el rendimiento para las operaciones de coma flotante trigonométricas, como puedan ser senos, cosenos, u otras más generales como la exponenciación, era muy pobre, lo que ha lastrado el resultado del procesador MicroBlaze. Este mismo problema se puede apreciar en el procesador ARM y en el MIPS.

Cabe destacar que el problema del procesador MIPS se solucionó haciendo uso de las instrucciones especiales de la unidad VFPU que integra^[54], capaz de realizar todo el conjunto de operaciones del benchmark Whetstone nativamente en el procesador. De esta manera el rendimiento se multiplicó en más de 28 veces, lo que demuestra la importancia de este tipo de unidades. También es necesario destacar que la VFPU sólo es capaz de trabajar con coma flotante simple, por lo que su uso para programas de coma flotante de precisión doble sería totalmente inútil.

El siguiente análisis es el rendimiento en términos de transferencias de datos para la memoria. Para ello, se van a comparar directamente todas las arquitecturas:

VitiRAM - Escritura

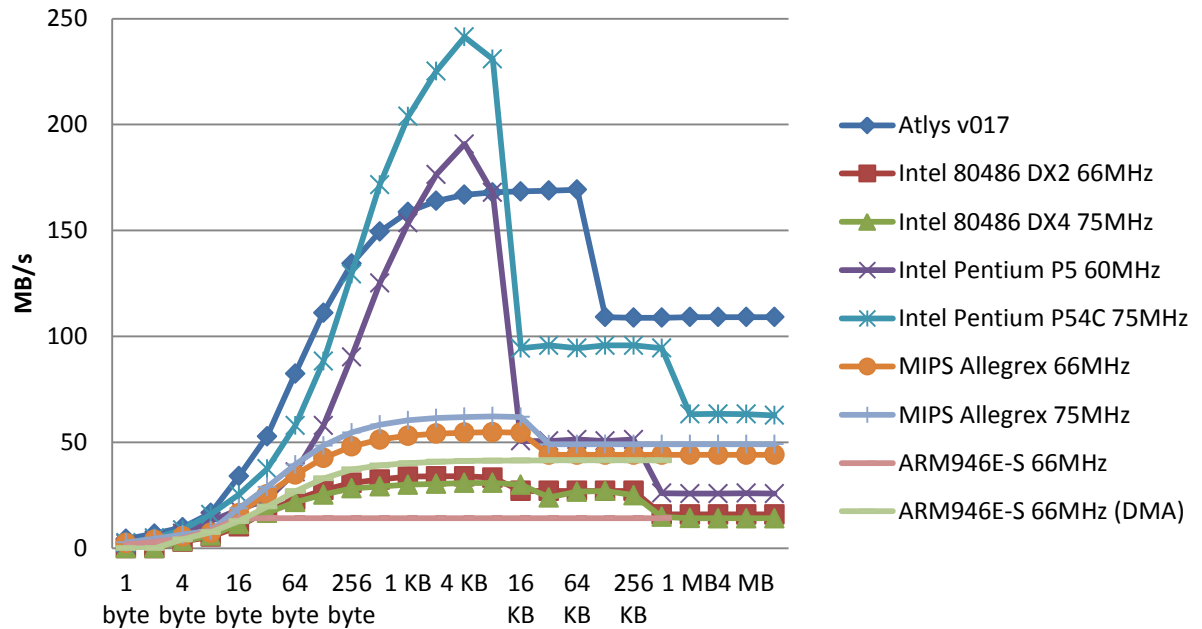


Ilustración 64: Resultados de VitiRAM en escritura para distintas arquitecturas

VitiRAM - Copia de datos

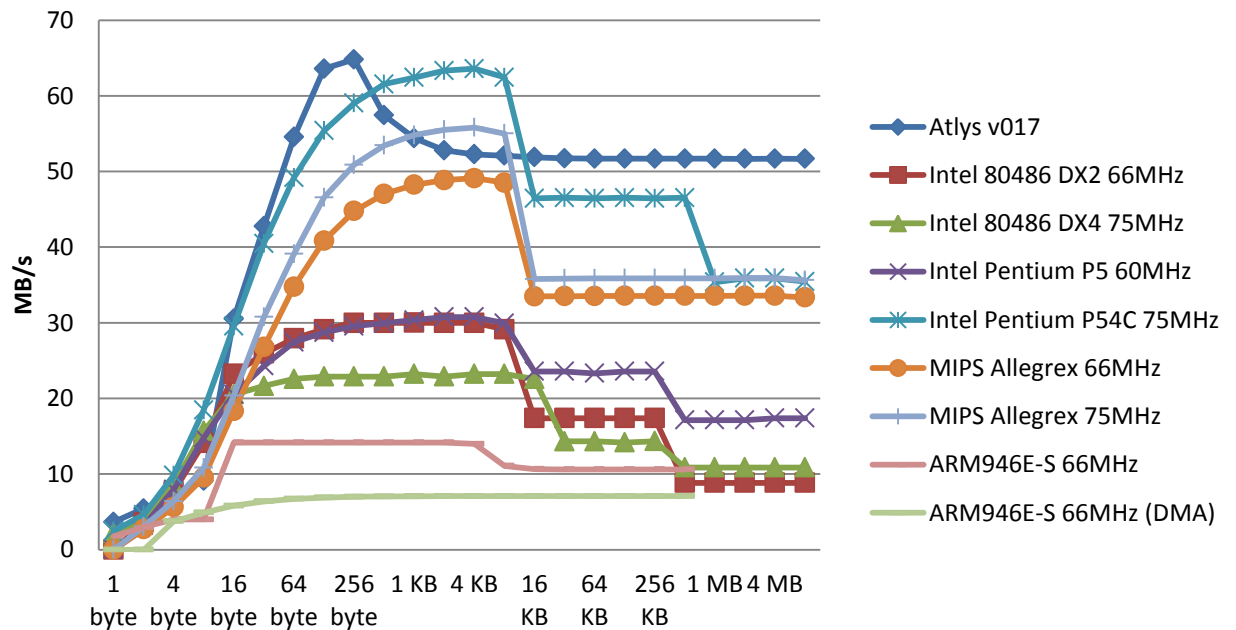


Ilustración 65: Resultados de VitiRAM en copias de datos para distintas arquitecturas

En las dos pruebas realizadas, se aprecia claramente cómo la memoria incorporada en la placa Digilent Atlys tiene un ancho de banda muy superior a las demás. La memoria DDR2 a 800MHz supera con creces a las memorias SDRAM, DRAM y cachés L2 usadas en los ordenadores con procesadores Intel Pentium e Intel 80486DX (éstas funcionan a la misma frecuencia que el bus de la placa base que utilizan, 33 MHz para el DX2, 25 MHz para el DX4, y 60 o 66 MHz para los Intel Pentium), y las memorias que incluyen el resto de plataformas (el tipo de memoria es desconocido para Nintendo DS y Sony PSP). También se puede apreciar como la caché de tipo L1 que incluye el procesador Intel Pentium esta más o menos a la par de los bloques de memoria BRAM que tiene la FPGA Spartan6.

Otro aspecto destacable, es el bajo rendimiento que posee el procesador ARM respecto a todos los demás, siendo éste el que peor resultados obtiene. Esta problemática se intentó solucionar utilizando el controlador de transferencias DMA que incorpora la Nintendo DS^[44], pero aun así no se consiguió una mejora suficiente. Según lo investigado, este problema se puede achacar a un fallo de diseño en la propia consola^[55].

Contrariamente, el diseño de la Sony PSP resulta mucho más eficiente, a pesar de utilizar un bus de datos que funciona a la mitad de frecuencia que el procesador, es decir, 33MHz en el bus de datos para una frecuencia de 66MHz en el procesador y 37MHz en el bus de datos para una frecuencia de 75MHz en el procesador. Esto mismo ocurre para los procesadores Intel, ya que su diseño se basa en multiplicadores respecto a la frecuencia base del bus de datos.

El último aspecto a analizar es el rendimiento de la predicción de saltos de todos los procesadores. Este es el resultado obtenido en ambas pruebas:

VitiJumps - Saltos directos

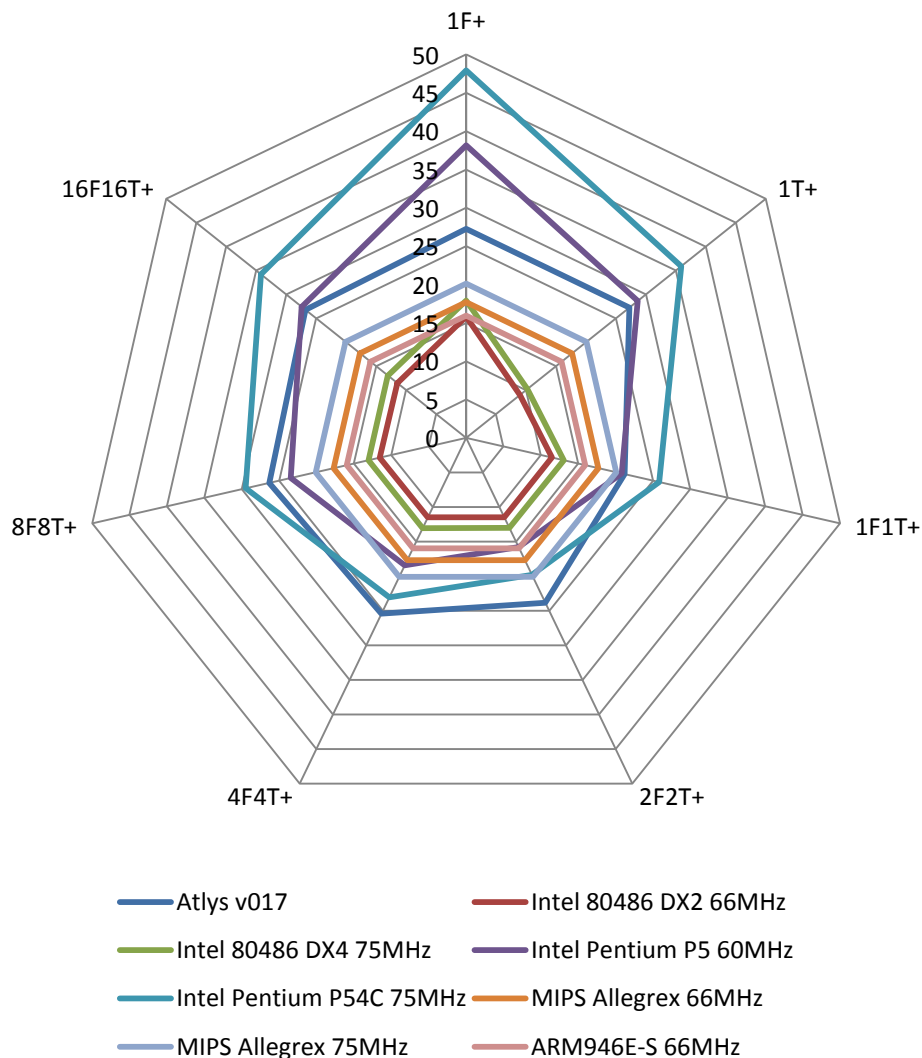


Ilustración 66: Resultados de VitiJumps en saltos directos para distintas arquitecturas

En esta gráfica se pueden apreciar diversos aspectos realmente interesantes:

- Los procesadores **Intel 80486DX** obtienen su mejor resultado cuando todas las condiciones de los saltos son negativas, y obtienen su peor resultado cuando todas las condiciones son positivas. Esto determina que el predictor de saltos que incluye es muy primitivo, el cual predice que todos los saltos son negativos. Es por ello que el resultado para las series de saltos positivos y negativos el resultado se mantiene constante.

- El procesador **Allegrex** tiene el mismo resultado para todas las condiciones, debido a la implementación de un *delay slot* para la primera instrucción que precede al salto. Esto implica que dicha instrucción se ejecuta previa e independientemente de lo que ocurra en el salto, mejorando en gran medida el resultado.
- El procesador **ARM946E-S**, al igual que el procesador Allegrex, obtiene el mismo resultado para todas las condiciones, pero su rendimiento es menor. El motivo es la ausencia de una unidad de predicción de saltos. La documentación que proporciona ARM indica que cada salto se realiza en un total de 3 ciclos de reloj, y que un fallo en el salto provoca un retardo de un ciclo adicional.
- Los procesadores **Intel Pentium P5/P54C** y **MicroBlaze** obtienen resultados muy parejos y con un buen rendimiento. El procesador Pentium incluye una unidad BTB (*branch target buffer*) capaz de predecir la dirección de un salto, con cuatro etapas distintas y un contador de dos bits para cada salto realizado (predictor bimodal, con 256 posibles entradas):
 - Salto con muy poca posibilidad de ocurrir
 - Salto con poca posibilidad de ocurrir
 - Salto que probablemente ocurrirá
 - Salto que muy probablemente ocurrirá

De esta manera, la predicción para los saltos continuados falsos o verdaderos del Pentium es superior al MicroBlaze, pero resulta ineficiente cuando se siguen rachas cortas de dos o cuatro saltos del mismo tipo. Por otro lado, el BTC (*branch target caché*) del procesador MicroBlaze funciona de manera parecida al Pentium, concretamente almacena los saltos realizados, pero aplica otra política de predicción de saltos (no está documentada por Xilinx), y obtiene resultados similares para todos los tipos de condiciones en los saltos.

El siguiente paso es analizar los resultados para los saltos indirectos, en la Ilustración 67 se pueden apreciar los resultados obtenidos:

VitiJumps - Saltos indirectos

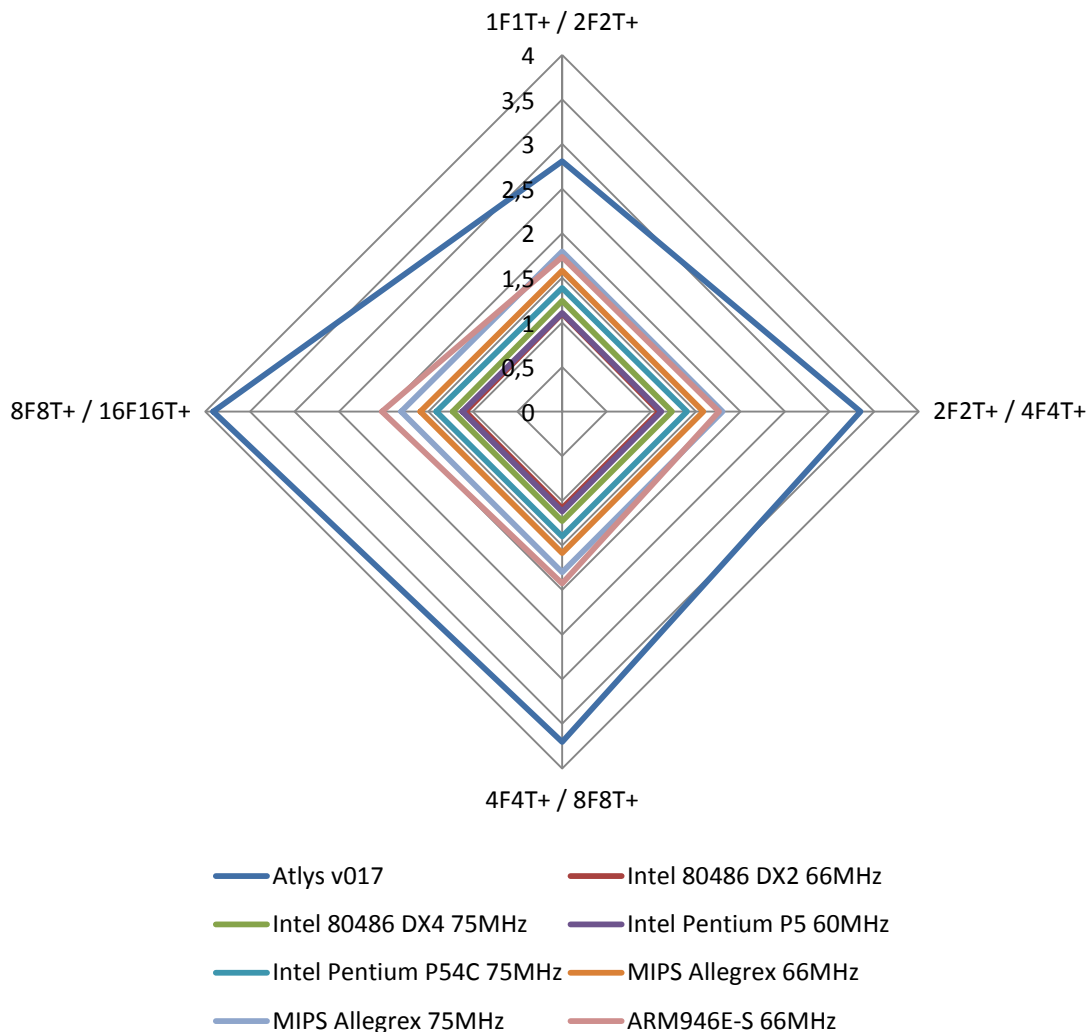


Ilustración 67: Resultados de VitiJumps en saltos indirectos para distintas arquitecturas

Los saltos indirectos resulta una prueba muy clara de la superioridad del predictor de saltos del procesador MicroBlaze respecto a los demás, llegando a obtener un rendimiento 50% superior a los demás procesadores probados. También se puede observar cómo el procesador ARM946E-S es el segundo mejor resultado junto con el procesador Allegrex.

Este resultado además es cuanto menos sorprendente, puesto que el procesador ARM como se comentó anteriormente carecía de unidad de predicción de saltos, lo que disminuiría el rendimiento respecto otras arquitecturas.

Ahora vamos a analizar el consumo eléctrico y el rendimiento. La placa Digilent Atlys es capaz de proporcionar datos de consumo eléctrico en mW, con una gran precisión (menos de un 1% de error) desde las fuentes de electricidad que utiliza. Las fuentes de electricidad son cuatro:

- **3.3 Voltios:** proporciona alimentación para la I/O de la FPGA, módulo de vídeo, puertos USB, relojes, ROM y códec de audio.
- **2.5 Voltios:** proporciona alimentación auxiliar a la FPGA, puerto de Digilent VHDC, I/O del puerto Ethernet y GPIO.
- **1.2 Voltios:** proporciona la alimentación al núcleo de la FPGA y el núcleo del módulo Ethernet.
- **1.8 Voltios:** alimentación para la I/O de la FPGA con la DDR y alimentación para la propia DDR.
- **0.9 Voltios:** voltaje de terminación de la DDR. Esta fuente no se puede medir, tiene un consumo máximo de 3 amperios y una media de 0.9 amperios (hasta 2.7 Vatios de consumo real).

Para medir el consumo del resto de sistemas, se utilizará la documentación técnica disponible y el medidor de consumo “Watts Up? .Net”. Concretamente, para los procesadores Intel se tomará el valor denominado TDP (Termal Disipation Power) que indica el mayor consumo en vatios que tiene el procesador, mientras que para las plataformas ARM y MIPS se utilizará el vatímetro en las consolas Nintendo DS y Sony PSP.

La medición con el vatímetro se realiza extrayendo la batería del sistema empotrado y con el cargador eléctrico conectado, de tal manera que se puede conocer el consumo exacto de los sistemas en cuestión.



Ilustración 68: Vatímetro Watts Up? .Net

Obtendremos los consumos energéticos en la ejecución de los benchmarks, con lo que podremos evaluar la eficiencia de la placa en términos de consumo y rendimiento por consumo. Primeramente, se ha analizado el consumo de la FPGA con el *bootrom* del XilKernel y la FPGA programadas. Los resultados fueron los siguientes para un lapso de tiempo de 5 segundos:

- 3.3 Voltios: consumo total de 2.316 Vatios de media
- 2.5 Voltios: consumo total de 0.1713 Vatios de media
- 1.8 Voltios: consumo total de 1.2692 Vatios de media
- 1.1 Voltios: consumo total de 0.6933 Vatios de media

Estos resultados muestran un consumo total de 4.4498 Vatios de media para toda la Digilent Atlys en estado de reposo. Para los benchmarks se utilizará el valor máximo en consumo por cada fuente de electricidad, lo que nos dará el peor caso posible para la plataforma. Por otro lado, los consumos máximos para el resto de arquitecturas son los siguientes:

- Intel 80486DX2 66MHz: 6.3 Vatios
- Intel 80486DX4 75MHz: 3.96 Vatios
- Intel Pentium P5 60MHz: 15.28 Vatios
- Intel Pentium P54C 75MHz: 9.54 Vatios
- Sony Playstation Portable: >0.1 Vatios
- Nintendo DS: >0.1 Vatios

Las medidas obtenidas por el vatímetro no han sido lo suficientemente precisas para el bajo consumo de estos dos últimos sistemas empotrados, por lo que se considerará el consumo de ambas como 0.1 Vatios.

Finalmente, aparte de la comparación energética de rendimiento por consumo, se ha añadido la comparación de rendimiento por megahercio. Esta comparación nos permitirá saber qué arquitectura es más eficiente por velocidad de reloj. Estos son los resultados energéticos obtenidos para los distintos benchmarks:

- **Dhrystone 2.1:** el consumo máximo obtenido para la Digilent Atlys ha sido 4.7481 Vatios utilizados, lo que nos da un resultado de 36.5325 DMIPS/Vatio.

Plataforma	Rendimiento energético (DMIPS/Vatio)	Rendimiento por MHz (DMIPS/MHz)
Atlys v017	36.53	2.62
Intel 80486DX2 66MHz	5.98	0.57
Intel 80486DX4 75MHz	10.8	0.57
Intel Pentium P5 60MHz	4.17	0.96
Intel Pentium P54C 75MHz	8.12	1.03
Nintendo DS ARM946E-S 66MHz	959	1.45
Sony PSP Allegrex 66MHz	741.8	1.12
Sony PSP Allegrex 75MHz	843	1.12

Tabla 38: Comparativa de rendimiento energético en Dhrystone 2.1

- **Whetstone:** el consumo máximo obtenido para la versión de precisión simple ha sido 4.8628 Vatios, mientras que para la versión de precisión doble ha sido 4.8643 Vatios. Esto nos da un resultado de 0.3898 MWIPS/Vatio y 0.275 MWIPS/Vatio.

Plataforma	Rendimiento energético (MWIPS/Vatio)	Rendimiento por MHz (MWIPS/MHz)
Atlys v017	SP: 0.38 – DP: 0.27	SP: 0.03 – DP: 0.02
Intel 80486DX2 66MHz	SP: 2.55 – DP: 2.35	SP: 0.24 – DP: 0.22
Intel 80486DX4 75MHz	SP: 4.72 – DP: 4.30	SP: 0.24 – DP: 0.22
Intel Pentium P5 60MHz	SP: 2.40 – DP: 1.99	SP: 0.61 – DP: 0.50
Intel Pentium P54C 75MHz	SP: 4.80 – DP: 3.99	SP: 0.61 – DP: 0.50
Nintendo DS ARM946E-S 66MHz	SP: 42.11 – DP: 37.08	SP: 0.06 – DP: 0.05
Sony PSP Allegrex 66MHz	SP: 266.95 – DP: 7.02	SP: 0.40 – DP: 0.01
Sony PSP Allegrex 75MHz	SP: 308.27 – DP: 7.98	SP: 0.41 – DP: 0.01

Tabla 39: Comparativa de rendimiento energético en Whetstone

- **Linpack:** se ha considerado la media de los resultados obtenidos para las versiones desenrolladas y sin desenrollar, en las versiones de precisión simple y precisión doble. Para la precisión simple el consumo máximo fue de 4.907 Vatios y para la precisión doble 4.9172 Vatios. Haciendo una media con los resultados obtenidos para este benchmark, obtenemos un rendimiento energético de 2.0725 MFlops/Vatio para la precisión simple, y 0.0589 MFlops/Vatio para la precisión doble. Como se puede observar, la diferencia de rendimiento energético entre la precisión simple y la precisión doble es abismal.

Plataforma	Rendimiento energético (MFlops/Vatio)	Rendimiento por MHz (MFlops/MHz)
Atlys v017	SP: 2.07 – DP 0.05	SP: 0.15 – DP: 0.004
Intel 80486DX2 66MHz	SP: 0.55 – DP: 0.49	SP: 0.05 – DP: 0.04
Intel 80486DX4 75MHz	SP: 0.95 – DP: 0.79	SP: 0.05 – DP: 0.04
Intel Pentium P5 60MHz	SP: 0.55 – DP: 0.49	SP: 0.14 – DP: 0.12
Intel Pentium P54C 75MHz	SP: 1.16 – DP: 1.08	SP: 0.14 – DP: 0.13
Nintendo DS ARM946E-S 66MHz	SP: 12.8 – DP: 7.5	SP: 0.02 – DP: 0.01
Sony PSP Allegrex 66MHz	SP: 62.50 – DP: 1.8	SP: 0.09 – DP: 0.002
Sony PSP Allegrex 75MHz	SP: 70.25 – DP: 2.0	SP: 0.09 – DP: 0.002

Tabla 40: Comparativa de rendimiento energético en Linpack

Los resultados obtenidos demuestran claramente que las plataformas que menos consumo eléctrico tienen son la Nintendo DS y la Sony PSP, y que a la vez ofrecen una mejor relación de potencia por vatio en prácticamente todos los benchmarks. Por otro lado la arquitectura MicroBlaze implementada en la FPGA Spartan6 consigue junto con la arquitectura x86 implementada en los procesadores Intel Pentium P5, el mejor rendimiento posible para los benchmarks y una mejor relación de rendimiento por megahertzio.

La idea principal que se puede extraer de estas pruebas constata que las FPGA de Xilinx junto con el soft-processor MicroBlaze ofrecen una potencia y eficiencia energética de término medio entre arquitecturas de bajo consumo, como puedan ser ARM o MIPS y arquitecturas más potentes como puedan ser x86 o PowerPC. Es destacable además que la placa de desarrollo cuenta con una gran cantidad de componentes que no han sido utilizados, como puedan ser el conector Ethernet, los puertos HDMI, los puertos USB o el códec de audio AC-97, y que han añadido un consumo extra que podría haber sido reducido.

En el siguiente apartado veremos la planificación seguida para el proyecto, y los costes totales de desarrollo.

7 Planificación y presupuesto

En este apartado se incluye una planificación detallada del proyecto, así como el presupuesto que ha costado realizar el mismo.

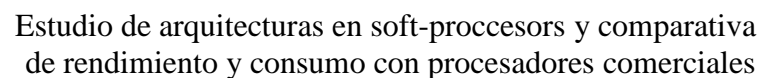
7.1 Planificación

La planificación llevada a cabo durante el desarrollo del proyecto se va a mostrar siguiendo primero una lista de tareas llevadas a cabo, y posteriormente un diagrama de Gantt más explicativo:

La duración del proyecto ha sido de aproximadamente 10 meses, siendo éste comenzado a mediados de Noviembre de 2011, y finalizado a comienzos de Octubre de 2012. A continuación se muestran las tareas realizadas junto con el tiempo llevado a cabo y el periodo que ha comprendido:

- **Propuesta del proyecto:** Tarea que consiste en el análisis de los distintos tipos de proyectos disponibles para FPGAs, y selección y refinamiento del aquí realizado.
- **Análisis:** Tareas de estudio y desglose de todos los aspectos a considerar sobre el proyecto, y definición de las características y requisitos del proyecto.
 - Definición del problema.
 - Definición de características generales.
 - Definición de requisitos.
 - Estudio de benchmarks.
- **Diseño:** Tareas de creación del diseño base para las FPGAs y los benchmarks, además de la creación del proceso iterativo para el proyecto.
 - Diseño de arquitecturas FPGA.
 - Diseño de benchmarks.
 - Diseño del proceso iterativo.

- **Implementación:** Tareas de creación de los benchmarks para el proyecto y adaptación para las distintas arquitecturas, y creación de todas las versiones de sistemas empotrados basados en FPGAs.
 - Adaptación de Dhrystone 2.1.
 - Adaptación de Whetstone.
 - Adaptación de Linpack 100x100.
 - Creación de VitiJumps.
 - Creación de VitiRAM.
 - Creación de plataformas Digilent Nexys3.
 - Creación de plataformas Digilent Atlys.
- **Evaluación:** Tareas de medición, comparación y estudio de los resultados obtenidos por los benchmarks en las FPGAs y procesadores comerciales utilizados.
 - Primera iteración.
 - Segunda iteración.
 - Tercera iteración.
 - Cuarta iteración.
- **Documentación:** Tarea de realización de este documento.



7.2 Presupuesto

En este apartado se va a realizar una estimación de los gastos que han supuesto la realización de este proyecto. El desglose del presupuesto se realiza en distintas categorías, con el objetivo de conseguir el mayor detalle posible en el mismo. Es importante destacar que todos los precios incluidos en el desglose no incluyen el IVA, siendo éste calculado con IVA al final en los costes totales del proyecto.

El desglose del presupuesto del proyecto es el siguiente:

- **Costes de personal**

En el proyecto sólo ha participado un ingeniero junior durante los 10 meses de duración del proyecto. Se han trabajado un total de 270 días, tal y como se ha podido observar en el diagrama de Gantt realizado en la planificación, y se ha estimado que se han trabajado durante 4 horas por día laborable aproximadamente.

El precio estimado por hora para cada tipo de persona involucrada en el proyecto es de 16 euros/hora. Este valor se ha estimado calculando la media de precios de contratación existentes para un ingeniero junior en la actualidad. El número de horas y el dinero asignado ha sido el siguiente:

Concepto	Salario/hora	Días laborables	Horas trabajadas al día	Total
Ingeniero junior	16,00 €/hora	270 días	4 horas/día	17.280,00€
				Total: 17.280,00€

Tabla 41: Costes de personal

- **Costes de hardware**

En este apartado se incluyen todos los elementos hardware que se han utilizado en el proyecto, desde el puesto de trabajo hasta los medidores de consumo eléctrico:

Concepto	Total
Ordenador portátil HP HDX16 1160ES	1.600€
Digilent Nexys3	151,79€
Digilent Atlys	266,26€
Sony Playstation Portable	209,95€
Nintendo DS	50,00€
Placa base y componentes para Intel 80486DX	50,00€
Placa base y componentes para Intel Pentium P5 y P54C	50,00€
Procesador Intel 80486DX2 66MHz	10,00€
Procesador Intel 80486DX4 75MHz	10,00€
Procesador Intel Pentium P5 60MHz	25,00€
Procesador Intel Pentium P54C 75MHz	7,00€
Vatímetro Watts up? .NET	180,01€
	Total: 2.610,01€

Tabla 42: Costes de hardware

Cabe destacar que los precios mostrados para las placas base, componentes y procesadores Intel 80486DX e Intel Pentium P5/P54C son los estipulados actualmente como piezas de segunda mano, puesto que en la actualidad éstos no se pueden adquirir como productos nuevos. También se ha de destacar que los precios de las placas de desarrollo Digilent son los precios estipulados para proyectos académicos, si fueran para uso no académico los precios serían aún mayores.

- **Costes de software**

Ahora vamos a calcular los precios utilizados en las licencias de software, aunque cabe destacar que en este proyecto se han utilizado licencias académicas, por lo que el precio real en este apartado sería de cero euros. Aun así, se ha calculado los precios de las licencias en el hipotético caso de tener que desarrollarlo para uso no académico:

Concepto	Total
Xilinx ISE Design Suite 13.3	2.861,41€
Microsoft Office Professional 2010	699,90€
	Total: 3.561,31€

Tabla 43: Costes de software

El resto de licencias software utilizadas son totalmente libres y gratuitas, por lo que se ha procedido a no incluir estos precios en el coste total.

- **Presupuesto final**

Finalmente, se calcula la suma de todas las categorías, y se agrega un tanto por ciento para los riesgos que puedan surgir en el proyecto, y un tanto por ciento para la obtención de beneficio con la realización del proyecto. El presupuesto final es por tanto, el siguiente:

Concepto	Total
Costes de personal	17.280,00€
Costes de hardware	2.610,01€
Costes de software	3.561,31€
Subtotal	23.451,32€
Riesgo (10%)	2.345,13€
Beneficio (33%)	7.738,94€
Total sin IVA	33.535,39€
	Total con IVA (21%): 40.577,82€

Tabla 44: Presupuesto final

El presupuesto total de ejecución de este proyecto asciende a la cantidad de **TREINTA Y TRES MIL QUINIENTOS TREINTA Y CINCO CON TREINTA Y NUEVE EUROS** sin incluir el impuesto sobre el valor añadido.

Incluyendo el IVA al 21% el presupuesto final es de **CUARENTA MIL QUINIENTOS SETENTA Y SIETE CON OCHENTA Y DOS EUROS (IVA 21%)**.

8 Conclusiones y trabajos futuros

En este apartado se resumen las conclusiones e ideas que se han obtenido en la realización de este proyecto, y posteriormente son analizadas las líneas de investigación futuras, con las cuales podríamos continuar, mejorar y avanzar este proyecto. Primeramente se va a observar desde un punto de vista técnico, donde veremos si se han logrado alcanzar las metas del proyecto, las tecnologías utilizadas y diversos aspectos técnicos, y posteriormente se va a observar desde un punto de vista personal.

8.1 Conclusiones

La principal conclusión que se puede obtener de este proyecto es que se ha conseguido obtener un diseño hardware de sistema empotrado completo, funcional y potente en base a un soft-processor implementado en una FPGA. El procesador MicroBlaze, implementado en la Spartan6 de Xilinx es capaz de ejecutar código simple o código con operaciones de coma flotante simple de manera eficiente, a la par de tener el rendimiento suficiente como para ejecutar aplicaciones de tamaño considerable.

Es cierto que el rendimiento ofrecido no es lo más potente que se pueda encontrar hoy en día comparado con ordenadores de escritorio o servidores, pero sí que es perfectamente válido para sistemas empotrados, como puedan ser dispositivos móviles, dispositivos industriales o cualquier tipo de electrodoméstico. Además, el paso de diseñar la plataforma hardware en la FPGA y ejecutar código dentro de ella no resulta excesivamente complicado, con el único inconveniente de los tiempos de compilación y sintetizado de los módulos VHDL.

Para ello, se ha conseguido diseñar y desarrollar pruebas de rendimiento capaces de evaluar el rendimiento de distintos aspectos de una arquitectura, siendo el código de dichos benchmarks reutilizable y adaptable a otras arquitecturas. Con estos benchmarks se ha evaluado el rendimiento de las unidades ALU, FPU y de predicción de saltos, además del ancho de banda disponible en la memoria RAM. Junto con la evaluación del rendimiento, también se ha podido evaluar el rendimiento energético, y contrastar todos estos resultados con otras plataformas y arquitecturas.

Por otro lado, cabe destacarse que el sistema operativo XilKernel, es problemático y en cierta medida difícil de utilizar, puesto que no todas las funciones de C/POSIX están correctamente implementadas, estando incluso algunas de ellas sin implementar. He aquí un ejemplo de lo comentado:

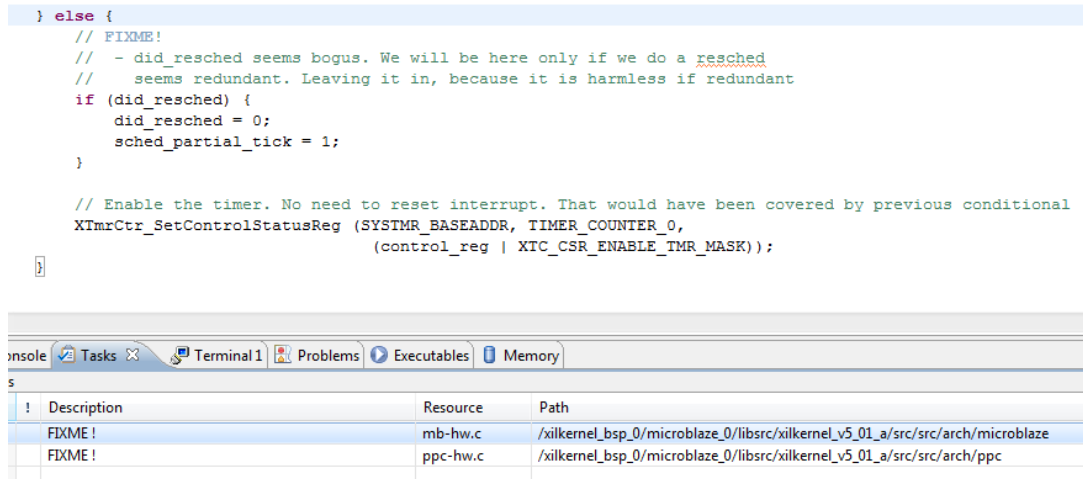


Ilustración 70: Implementación dudosa del XilKernel

Además, el proceso de debugging de las aplicaciones no resultó sencillo, puesto que se producían fallos de origen desconocido, incluso en el modo de desarrollo del Xilinx SDK.

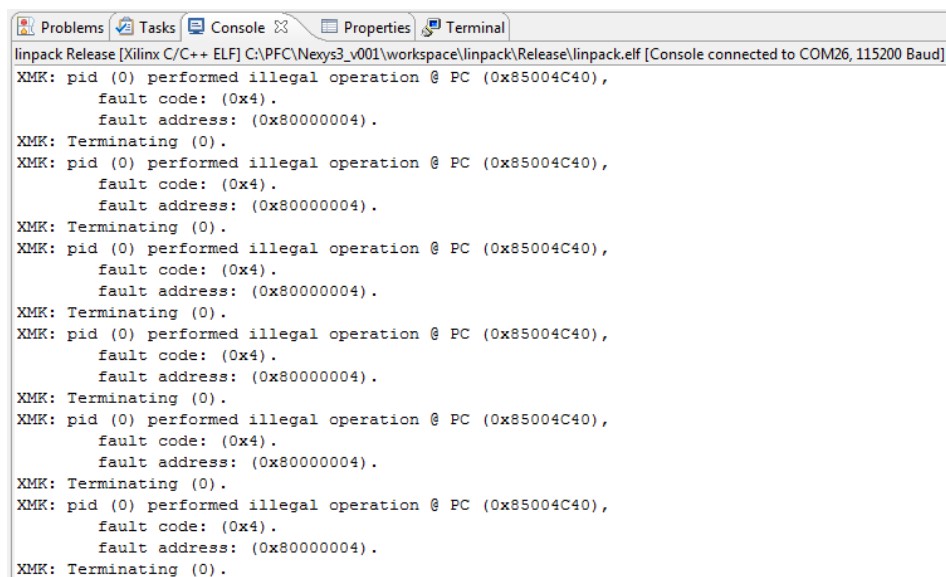


Ilustración 71: Error grave de ejecución en el XilKernel

Un aspecto que no se ha podido realizar en el desarrollo en el proyecto es la implementación de coprocesadores que acelerasen las operaciones que el procesador no pudiese realizar eficientemente, como son las operaciones en coma flotante de precisión doble. El motivo principal fue que en ningún sitio se encontraron unidades FPU o VFPU implementadas para el bus FSL del MicroBlaze, y su implementación manual requiere de un gran equipo de desarrolladores bien conocedores de los lenguajes VHDL y de las FPGAs.

En cuanto a las conclusiones personales, este proyecto me ha servido para aprender a implementar soft-processors en FPGAs, y sobre todo he aprendido cómo se han de analizar, diseñar e implementar pruebas de rendimiento para equipos. La implementación de los soft-processors ha servido como solución real para sistemas empujados, por lo que este aspecto es para mí muy importante, enseñándome qué aspectos considerar como prioritarios (consumo energético, potencia, utilidad, elementos en la arquitectura, etc.).

La implementación de las pruebas de rendimiento me ha enseñado a plantear de distinta manera el desarrollo de diseños software, en este caso no se ha de centrar en lo bonito o reutilizable del código, sino en la sencillez del mismo, cuanto menos añadidos y extras tenga mejor. La evaluación, por otro lado, me ha demostrado que no por ofrecer características mejores significa que vayas a obtener un resultado mejor.

Me gustaría puntualizar también que la realización este documento me ha servido para enfocar los proyectos desde otro punto de vista, pudiendo analizar y comprender los fallos y puntos faltantes respecto a las implementaciones de código con los requisitos establecidos para el proyecto.

8.2 Líneas de investigación futuras

El diseño y evaluación realizados en este proyecto para la implementación del mejor soft-processor posible como sistema empujado puede ser ampliada de diversas maneras, dado que en este caso solo hemos utilizado el procesador MicroBlaze. Este es un listado de las posibles líneas de investigación futuras:

1. **Cambiar el tipo de bus de datos:** El bus de datos utilizado en este proyecto es el denominado PLB (*processor local bus*). Este es el recomendado por Digilent para sus placas, pero es posible la utilización de otro bus de datos, concretamente el bus AXI (*advanced extensible interface*). La diferencia principal entre ambos buses reside en que PLB funciona en modo *big-endian* mientras que AXI funciona en modo *little-endian*. Esto implica que el procesador también ha de cambiar en su modo de funcionamiento, lo que podría ocasionar que el rendimiento del MicroBlaze fuese distinto al analizado en este proyecto. Cabe destacarse que el bus AXI es más actual y avanzado que el bus PLB, y es desarrollado por la empresa ARM.

2. **Evaluación de rendimiento en Linux:** Todas las pruebas de rendimiento para el MicroBlaze se han realizado bajo el sistema operativo mínimo y propio de Xilinx, el XilKernel. Este kernel implementa la funcionalidad POSIX parcialmente, lo que podría ocasionar que el rendimiento del procesador fuera mayor que con un sistema operativo más complejo. La opción más real sería la utilización del sistema operativo Linux, puesto que es compatible con el procesador MicroBlaze. Una de las opciones principales es la distribución PetaLinux, específicamente diseñada para su implementación en FPGAs de Xilinx.
3. **Implementación en FPGAs de mayor rendimiento:** Las placas de desarrollo utilizadas, la Digilent Nexys3 y la Digilent Atlys, hacen uso de la FPGA Spartan6. Este es el modelo más básico que proporciona Xilinx, teniendo gamas más altas, como son las series 7 (Artix, Kintex y Virtex). Estas FPGAs poseen una mayor cantidad de LUTs y biestables disponibles, a la par que una mayor *BlockRAM*. Y no solamente son de mayor tamaño, sino que permiten frecuencias mucho mayores, siendo el máximo unos 233 MHz para el procesador MicroBlaze en la familia Virtex6.
4. **Implementación de coprocesadores matemáticos:** En este proyecto no se ha podido incluir coprocesadores matemáticos capaces de incrementar el rendimiento del procesador MicroBlaze a través del bus específico FSL (*fast simple link*). Como se ha podido observar en el proyecto, la utilización de la unidad VFPU en la Sony Playstation Portable ha permitido incrementar su rendimiento en gran medida, por lo que su implementación dentro de la FPGA también sería de gran ayuda. Además se podrían solventar problemas como el bajo rendimiento en coma flotante de precisión doble.
5. **Realización de más pruebas de rendimiento:** Tal y como se vio en el estado del arte, existen gran cantidad de benchmarks para la evaluación de los distintos aspectos de una plataforma empotrada. Además se podrían evaluar otros apartados aparte de los analizados en este proyecto, como pudiera ser el rendimiento en operaciones de transferencias de datos en red, o el rendimiento en el tratamiento de señales de video y de audio.
6. **Utilización de versiones actualizadas del Xilinx ISE:** La versión utilizada en este proyecto del Xilinx ISE es la 13.3, siendo ésta la recomendada por Digilent. Sin embargo, no es la versión más nueva del producto, la versión actual es la 14.1, y en ella han conseguido optimizar los procesos de routing y mapping del hardware en la FPGA, además de realizar optimizaciones en las interrupciones del procesador MicroBlaze. Estas dos mejoras harían que el rendimiento fuese aún mayor.
7. **Evaluación de otros tipos de soft-processors:** En este proyecto nos hemos centrado únicamente en el procesador MicroBlaze, pero existen gran cantidad de soft-processors alternativos a éste. Aparte de los vistos en el estado del arte, existen otras arquitecturas aún más potentes implementables en FPGAs, como pueden ser los soft-processors OpenSparc T1 y T2 (y la variante de menor tamaño S1 Core).

8. **Comparación de los resultados obtenidos con otros sistemas empotrados:** En este proyecto todos los resultados del MicroBlaze se han comparado con procesadores de un rendimiento similar y arquitecturas distintas. Sin embargo, existen otras arquitecturas y sistemas empotrados de distinta potencia con los que se podrían comparar los resultados. Un claro ejemplo y fácil de conseguir sería la implementación de los benchmarks en la Raspberry Pi (sistema empotrado de muy bajo consumo, precio reducido y potencia computacional media), o en dispositivos móviles basados en los sistemas operativos Android o iOS.
9. **Orientación del proyecto a la evolución de las FPGAs:** Las FPGAs han evolucionado de distintas formas, y una de ellas es el acoplamiento de microprocesadores reales a las FPGAs, con el objetivo de obtener un rendimiento mucho mayor. Esto surge de la falta de potencia en general de los soft-processors, tal y como se ha podido observar en el proyecto. De esta manera se soluciona los cuellos de botella que se puedan ocasionar, a la par de simplificar el proceso de generación del hardware. Un claro ejemplo de esta tecnología es el Zynq 7000 de Xilinx, diseño que incluye un procesador ARM Cortex A9 de doble núcleo a una frecuencia de 1 GHz, el cual puede ser potenciado mediante coprocesadores implementados en la FPGA que incorpora.
10. **Optimización manual de los benchmarks realizados:** Los benchmarks utilizados en este proyecto han sido escritos intrínsecamente en el lenguaje de alto nivel C o C++. Por otro lado, sería posible realizar una optimización para cada arquitectura, incluyendo código en lenguaje ensamblador especializado en las secciones de código críticas del benchmark. De esta manera se podrían obtener mejores resultados para cada arquitectura, aunque el esfuerzo que conlleva esta optimización sería realmente elevado.



9 Acrónimos y abreviaturas

- **AES**: Advanced Encryption Standard
- **AHB**: Advanced High-performance Bus
- **ALU**: Arithmetic Logic Unit
- **AMBA**: Advanced Microcontroller Bus Architecture
- **AMD**: Advanced Micro Devices, Inc.
- **arctan**: arcotangente
- **ARM**: Advanced RISC Machine
- **ASIC**: Application-Specific Integrated Circuit
- **AXI**: Advanced eXtensible Interface
- **BLAS**: Basic Linear Algebra Subprograms
- **BRAM**: Block RAM
- **BSP**: Board Support Package
- **BTB**: Branch Target Buffer
- **BTC**: Branch Target Caché
- **CAS**: Column Address Strobe
- **CISC**: Complex Instruction Set Computing
- **cos**: coseno
- **CPI**: Cycles Per Instruction
- **CPLD**: Complex Programmable Logic Device
- **CRC**: Cyclic Redundancy Check
- **DDR**: Double Data Rate
- **DMA**: Direct Memory Access
- **DMIPS**: Dhrystone Million Instructions Per Second
- **DOS**: Disk Operating System
- **DP**: Double Precision
- **DRAM**: Dynamic Random-Access Memory
- **DSP**: Digital Signal Processor
- **DTLB**: Dual Translation Lookaside Buffer
- **EDK**: Embedded Development Kit
- **EEMBC**: Embedded Microprocessor Benchmark Consortium
- **ESA**: European Space Agency
- **exp**: exponencial
- **FPGA**: Field-Programmable Gate Array
- **FPU**: Floating-Point Unit
- **FSL**: Fast Simple Link
- **Gb/s**: gigabytes por segundo
- **GCC**: GNU Compiler Collection
- **GPIO**: General Purpose Input/Output

- **GPL:** GNU General Public License
- **HDL:** Hardware Description Language
- **HDMI:** High-Definition Multimedia Interface
- **HID:** Human Interface Device
- **IA-32:** Intel Architecture 32-bit
- **IDE:** Integrated Development Environment
- **IEEE:** Institute of Electrical and Electronics Engineers
- **IO:** Input/Output
- **IP core:** Intellectual Property Core
- **IPv4:** Internet Protocol versión 4
- **IPv6:** Internet Protocol versión 6
- **ISO:** International Organization for Standarization
- **JTAG:** Joint Test Action Group
- **Kb:** kilobyte
- **LE:** Logic Element
- **LGPL:** GNU Lesser General Public License
- **LLVM:** Low Level Virtual Machine
- **LMB:** Local Memory Bus
- **ln:** logaritmo neperiano
- **LRR:** Least Recently Replaced
- **LRU:** Least Recently Used
- **LUT:** LookUp Table
- **MAC:** Multiply-Accumulate Operation
- **MB:** megabyte
- **MB/s:** megabytes por segundo
- **MDMX:** MIPS Digital Media eXtension
- **MFlops:** Million Floating Operations Per Second
- **MHz:** megahertzio
- **MIPS:** Microprocessor with Interlocked Pipeline Stages
- **MPSoC:** MultiProcessor System-On-Chip
- **MVC:** Modelo Vista Controlador
- **mW:** milivatio
- **MWIPS:** Million Whestone Instructions Per Second
- **ns:** nanosegundo
- **ORPSoC:** OpenRISC Processor System-on-a-Chip
- **PAL:** Programmable Array Logic
- **PDA:** Personal Digital Assistant
- **PLB:** Processor Local Bus
- **POSIX:** Portable Operating System Interface
- **PSP:** Playstation Portable
- **PSRAM:** Pseudo-Static Random Access Memory

- **RAM:** Random Access Memory
- **RF:** radio-frequency
- **RISC:** Reduced Instruction Set Computing
- **ROM:** Read-Only Memory
- **RTC:** Real-Time Clock
- **RTL:** Register Transfer Level
- **RUC:** Requisito de capacidad
- **RUR:** Requisito de restriccion
- **SDK:** Software Development Kit
- **SDRAM:** Synchronous Dynamic Random Access Memory
- **SGI:** Silicon Graphics, Inc.
- **SIMD:** Single instruction, multiple data
- **sin:** seno
- **SMP:** Symmetric Multi-Processor
- **SoC:** System-On-Chip
- **SP:** Single Precision
- **SPARC:** Scalable Processor Architecture
- **sqrt:** raíz cuadrada
- **SRAM:** Static Random Access Memory
- **SSE:** Streaming SIMD Extensions
- **TCM:** Tightly-Coupled Memory
- **TCP:** Transmission Control Protocol
- **TDP:** Thermal Design Power
- **TLB:** Translation Lookaside Buffer
- **UART:** Universal Asynchronous Receiver/Transmitter
- **UDP:** User Datagram Protocol
- **USB:** Universal Serial Bus
- **VAX:** Virtual Address eXtension
- **VFP:** Vector Floating-Point
- **VFPU:** Vector Floating-Point Unit
- **VGA:** Video Graphics Array
- **VHDL:** VHSIC Hardware Description Lenguaje
- **WB:** Write-Back
- **WT:** Write-Through
- **XPS:** Xilinx Platform Studio
- **XSDK:** Xilinx Software Development Kit



10 Bibliografía

- [1] Enciclopedia sobre Field Programmable Gate Array (Español). 2012. Disponible:
<http://es.wikipedia.org/wiki/FPGA>
- [2] Enciclopedia sobre Field Programmable Gate Array (Inglés). 2012. Disponible:
<http://en.wikipedia.org/wiki/FPGA>
- [3] Productos y especificaciones de FPGAs de Xilinx. 2012. Disponible:
<http://www.xilinx.com/products/silicon-devices/fpga/index.htm>
- [4] Productos y especificaciones de herramientas de diseño de Xilinx. 2012. Disponible:
<http://www.xilinx.com/products/design-tools/index.htm>
- [5] Enciclopedia sobre Xilinx (Inglés). 2012. Disponible:
<http://en.wikipedia.org/wiki/Xilinx>
- [6] Productos y especificaciones de FPGAs de Altera. 2012. Disponible:
<http://www.altera.com/devices/fpga/fpga-index.html>
- [7] Productos y especificaciones de herramientas de diseño de Altera. 2012. Disponible:
<http://www.altera.com/products/software/sfw-index.jsp>
- [8] MicroBlaze Processor Reference Guide. 2012. Disponible:
http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_1/mb_ref_guide.pdf
- [9] Nios II Processor Reference Handbook. 2012. Disponible:
http://www.altera.com/literature/hb/nios2/n2cpu_nii5v1.pdf
- [10] Leon4 Processor. 2012. Disponible:
http://www.gaisler.com/cms/index.php?option=com_content&task=view&id=338&Itemid=231
- [11] OpenRISC 1200 IP Core Specification (Preliminary Draft). 2012. Disponible:
<http://openrisc.net/or1200-spec.html>
- [12] Enciclopedia sobre x86 (Inglés). 2012. Disponible:
<http://en.wikipedia.org/wiki/x86>
- [13] Embedded IntelDX2 Processor. 1995. Disponible:
<http://datasheets.chipdb.org/Intel/x86/486/datashts/27277001.pdf>
- [14] Embedded Write-Back Enhanced IntelDX4 Processor. 1995. Disponible:
<http://datasheets.chipdb.org/Intel/x86/486/datashts/27277101.pdf>
- [15] Intel486 DX Microprocessor. 1993. Disponible:
<http://datasheets.chipdb.org/Intel/x86/486/datashts/240440-006.pdf>
- [16] Pentium® Processor P5 Family Developer's Manual. 1995. Disponible:
http://datasheets.chipdb.org/Intel/x86/Pentium/241428_4.PDF
- [17] Pentium® Processor P54C. 1996. Disponible:
<http://datasheets.chipdb.org/Intel/x86/Pentium/24199708.PDF>
- [18] Pentium® Processor P54C with voltage reduction technology. 1996. Disponible:
<http://datasheets.chipdb.org/Intel/x86/Pentium/24297301.PDF>
- [19] Pentium® Processor P5. 1994. Disponible:

- <http://datasheets.chipdb.org/Intel/x86/Pentium/24159502.pdf>
- [20] Enciclopedia sobre arquitectura MIPS (Inglés). 2012. Disponible:
http://en.wikipedia.org/wiki/MIPS_architecture
- [21] MIPS32® Architecture. 2012. Disponible:
<http://www.mips.com/products/architectures/mips32/>
- [22] MIPS64® Architecture. 2012. Disponible:
<http://www.mips.com/products/architectures/mips64/>
- [23] MIPS R4000 Microprocessor User's Manual (Second Edition). 1993. Disponible:
http://hitmen.c02.at/files/docs/psp/R4400_Uman_book_Ed2.pdf
- [24] NEC Vr5432 64-Bit MIPS RISC Microprocessor. 2000. Disponible:
http://hitmen.c02.at/files/docs/psp/1375_V2.pdf
- [25] Enciclopedia sobre ARM (Inglés). 2012. Disponible:
<http://es.wikipedia.org/wiki/ARM>
- [26] Procesadores y especificaciones de ARM. 2012. Disponible:
<http://www.arm.com/products/processors/index.php>
- [27] ARM ARM946E-S (Rev1) System-on-Chip DSP enhanced processor (Product Overview). 2000. Disponible:
<http://infocenter.arm.com/help/topic/com.arm.doc.dvi0022a/DVI0022A.pdf>
- [28] ARM ARM946E-S (r1p1) Technical Reference Manual. 2007. Disponible:
http://infocenter.arm.com/help/topic/com.arm.doc.ddi0201d/DDI0201D_arm946es_r1p1_trm.pdf
- [29] Digilent Atlys™ Board Reference Manual. 2011. Disponible:
http://www.digilentinc.com/Data/Products/ATLYS/Atlys_rm.pdf
- [30] Digilent Nexys3™ Board Reference Manual. 2011. Disponible:
http://www.digilentinc.com/Data/Products/NEXYS3/Nexys3_rm.pdf
- [31] CellularRAM® (PSRAM) Memory. 2011. Disponible:
http://www.micron.com/~media/Documents/Products/Product%20Flyer/psram_flyer.pdf
- [32] Xilinx Spartan-6 Family Overview. 2011. Disponible:
http://www.xilinx.com/support/documentation/data_sheets/ds160.pdf
- [33] Performance of Various Computers Using Standard Linear Equations Software. 2011. Disponible:
<ftp://www.netlib.org/benchmark/performance.pdf>
- [34] The LINPACK Benchmark: Past, Present and Future. 2001. Disponible:
<http://www.netlib.org/utk/people/JackDongarra/PAPERS/hpl.pdf>
- [35] A synthetic benchmark. 1976. Disponible:
<http://www.roylongbottom.org.uk/whetstone.pdf>
- [36] Dhrystone: A synthetic systems programming benchmark. 1984. Disponible:
<http://www.cse.hcmut.edu.vn/~anhvu/teaching/2012/ACA/Dhrystone.pdf>
- [37] Understanding Variations in Dhrystone Performance. 1989. Disponible:
<https://www.cs.drexel.edu/~jjohnson/fa00/cs570/programs/dhrystone/VARIATIONS>
- [38] Dhrystone Benchmark (Pascal Version 2): Rationale and Measurement Rules. 1988. Disponible:

- <http://www.netlib.org/benchmark/dhry-pascal>
- [39] Fast and slow if-statements: branch prediction in modern processors. 2012. Disponible:
<http://igoro.com/archive/fast-and-slow-if-statements-branch-prediction-in-modern-processors/>
- [40] Performance Metrics: Measuring and quantifying computer systems performance. 2012. Disponible:
<http://www.cl.cam.ac.uk/teaching/0607/AdvSysTop/perf1.pdf>
- [41] Lecture 4: Benchmarks and Performance Metrics. 1996. Disponible:
<http://bnrg.eecs.berkeley.edu/~randy/Courses/CS252.S96/Lecture04.pdf>
- [42] Roy Longbottom's PC Benchmark Collection. 2012. Disponible:
<http://www.roylongbottom.org.uk>
- [43] Enciclopedia sobre Nintendo DS (Inglés). 2012. Disponible:
http://en.wikipedia.org/wiki/Nintendo_DS
- [44] DMA vs. ARM9 – fight!. 2009. Disponible:
<http://www.coranac.com/2009/05/dma-vs-arm9-fight/>
- [45] Enciclopedia sobre Playstation Portable (Inglés). 2012. Disponible:
http://en.wikipedia.org/wiki/Sony_Playstation_Portable
- [46] PSP Chip Block Diagram. 2012. Disponible:
<http://s2.subirimagenes.com/imagen/4478905diagrama-de-bloques.png>
- [47] NDS Tutorial Day 2: NDS Introduction. 2012. Disponible:
http://dev-scene.com/NDS/Tutorials_Day_2
- [48] PSP Programming. 2012. Disponible:
http://en.wikibooks.org/wiki/PSP_Programming
- [49] Open Watcom C/C++ Getting Started. 2008. Disponible:
http://www.openwatcom.org/ftp/manuals/current/c_readme.pdf
- [50] Options that Control Optimization (GCC). 2012. Disponible:
<http://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html>
- [51] Guide to applying the ESA software engineering standards to small software projects. 1996. Disponible:
<http://www.ie.inf.uc3m.es/grupo/docencia/reglada/Is1y2/ESA/BSSC962.PDF>
- [52] Evaluation of synthesizable CPU cores. 2004. Disponible:
http://www.gaisler.com/doc/Evaluation_of_synthesizable_CPU_cores.pdf
- [53] Comparison of Synthesizable Processor Cores. 2005. Disponible:
http://web.cecs.pdx.edu/~mperkows/CLASS_573/573_2007/CoSCPU.pdf
- [54] Quirks of PSP CPU disassembly. 2011. Disponible:
<http://code.google.com/p/pops-gte/wiki/DisasmHints>
- [55] GBATEK – GameBoy Advance / Nintendo DS technical info. 2011. Disponible:
<http://nocash.emubase.de/gbatek.htm>
- [56] Enciclopedia sobre el procesador Intel Atom (system on chip). 2012. Disponible:
[http://en.wikipedia.org/wiki/Atom_\(system_on_chip\)](http://en.wikipedia.org/wiki/Atom_(system_on_chip))



Anexo A: Material entregado

El cometido de este apartado es recoger todos los contenidos que se han incluido en el DVD del proyecto, diseccionando la estructura del mismo en carpetas y ficheros, y explicando el contenido de cada uno de ellos:

- **Documentación:** en esta carpeta se encuentra toda la documentación del proyecto, incluyendo esta memoria y la presentación realizada para el mismo.
 - **Memoria.pdf:** el presente documento en formato PDF.
 - **Presentacion.pptx:** presentación del proyecto en formato PowerPoint.
- **Resultados:** en esta carpeta se encuentran todos los resultados obtenidos, tanto en términos de potencia como de consumo.
 - **Resultados.xlsx:** libro de Excel con todos los resultados obtenidos.
- **Benchmarks:** esta carpeta recoge todos los benchmarks migrados para las distintas arquitecturas. En cada carpeta designada para las distintas arquitecturas, se encuentran los benchmarks Dhrystone 2.1, Whetstone, Linpack, VitiJumps y VitiRAM.
 - **PSP:** benchmarks adaptados para la arquitectura MIPS de la Sony Playstation Portable.
 - **NDS:** benchmarks adaptados para la arquitectura ARM de la Nintendo DS.
 - **PC:** benchmarks adaptados para las arquitectura x86 y x86-64.
 - **FPGA:** benchmarks adaptados para la arquitectura MicroBlaze y el sistema operativo XilKernel.
- **Arquitecturas:** en esta carpeta se encuentran los diseños de los sistemas empuados basados en el MicroBlaze para FPGAs. Todos los proyectos se han de abrir con el software Xilinx ISE 13.3.
 - **Nexys3:** proyectos de XPS con los diseños de las placas Digilent Nexys3.
 - **Atlys:** proyectos de XPS con los diseños de las placas Digilent Atlys.
- **Leeme.txt:** fichero explicativo de la información básica del proyecto y de la estructura del proyecto dentro del DVD



Anexo B: Resumen de arquitecturas implementadas

En este anexo se incluye una breve tabla/resumen que incluye todos los diseños implementados para las placas Digilent Nexys3 y Digilent Atlys, mostrándose los nombres de versión de cada una, y las características que tienen habilitadas o deshabilitadas.

Plataforma	Frecuencia (MHz)	FPU	Multiplicador ALU	Divisor ALU	Barrel Shifter	Bits adicionales en MSR	Pattern Comparator	BTC
Nexys3 v001	66.66	No	No	No	No	No	No	No
Nexys3 v002	66.66	No	No	No	Si	No	No	No
Nexys3 v003	66.66	No	No	No	Si	Si	No	No
Nexys3 v004	66.66	No	No	No	Si	Si	Si	No
Nexys3 v005	66.66	No	32 bits	No	Si	Si	Si	No
Nexys3 v006	66.66	No	32 bits	32 bits	Si	Si	Si	No
Nexys3 v007	66.66	No	64 bits	32 bits	Si	Si	Si	No
Nexys3 v008	66.66	No	64 bits	32 bits	Si	Si	Si	256
Nexys3 v009	66.66	Básica	64 bits	32 bits	Si	Si	Si	256
Nexys3 v010	66.66	Extendida	64 bits	32 bits	Si	Si	Si	256
Nexys3 v011	66.66	Extendida	64 bits	32 bits	Si	Si	Si	256
Nexys3 v012	66.66	Extendida	64 bits	32 bits	Si	Si	Si	256
Nexys3 v013	66.66	Extendida	64 bits	32 bits	Si	Si	Si	256
Nexys3 v014	66.66	Extendida	64 bits	32 bits	Si	Si	Si	256
Nexys3 v015	66.66	Extendida	64 bits	32 bits	Si	Si	Si	256
Nexys3 v016	66.66	Extendida	64 bits	32 bits	Si	Si	Si	256
Nexys3 v017	66.66	Extendida	64 bits	32 bits	Si	Si	Si	256
Nexys3 v018	66.66	Extendida	64 bits	32 bits	Si	Si	Si	256
Nexys3 v019	66.66	Extendida	64 bits	32 bits	Si	Si	Si	256

Plataforma	Frecuencia (MHz)	FPU	Multiplicador ALU	Divisor ALU	Barrel Shifter	Bits adicionales en MSR	Pattern Comparator	BTC
Nexys3 v024	50.00	Extendida	64 bits	32 bits	Si	Si	Si	256
Nexys3 v025	62.50	Extendida	64 bits	32 bits	Si	Si	Si	256
Nexys3 v026	75.00	Extendida	64 bits	32 bits	Si	Si	Si	256
Nexys3 v027	83.33	Extendida	64 bits	32 bits	Si	Si	Si	256

Tabla 45: Resumen de configuraciones hardware en Digilent Nexys3

Plataforma	Frecuencia (MHz)	FPU	Multiplicador ALU	Divisor ALU	Barrel Shifter	Bits adicionales en MSR	Pattern Comparator	BTC
Atlys v001	66.66	Extendida	64 bits	32 bits	Si	Si	Si	256
Atlys v002	66.66	Extendida	64 bits	32 bits	Si	Si	Si	256
Atlys v003	66.66	Extendida	64 bits	32 bits	Si	Si	Si	256
Atlys v007	66.66	Extendida	64 bits	32 bits	Si	Si	Si	256
Atlys v008	66.66	Extendida	64 bits	32 bits	Si	Si	Si	No
Atlys v009	66.66	Extendida	64 bits	32 bits	Si	Si	Si	2048
Atlys v010	66.66	Extendida	64 bits	32 bits	Si	Si	Si	64
Atlys v017	66.66	Extendida	64 bits	32 bits	Si	Si	Si	2048

Tabla 46: Resumen de configuraciones hardware en Digilent Atlys

Plataforma	Caché de datos	Caché de instrucciones	Política de reemplazo	Tamaño de palabra	Caché Stream	Caché Victim
Nexys3 v001	No	No	No	No	No	No
Nexys3 v002	No	No	No	No	No	No
Nexys3 v003	No	No	No	No	No	No
Nexys3 v004	No	No	No	No	No	No
Nexys3 v005	No	No	No	No	No	No
Nexys3 v006	No	No	No	No	No	No
Nexys3 v007	No	No	No	No	No	No
Nexys3 v008	No	No	No	No	No	No
Nexys3 v009	No	No	No	No	No	No
Nexys3 v010	No	No	No	No	No	No
Nexys3 v011	64 bytes	64 bytes	WT	4	No	No
Nexys3 v012	128 bytes	128 bytes	WT	4	No	No
Nexys3 v013	256 bytes	256 bytes	WT	4	No	No
Nexys3 v014	512 bytes	512 bytes	WT	4	No	No
Nexys3 v015	1 Kb	1 b	WT	4	No	No
Nexys3 v016	2 Kb	2 Kb	WT	4	No	No
Nexys3 v017	4 Kb	4 Kb	WT	4	No	No
Nexys3 v018	8 Kb	8 Kb	WT	4	No	No
Nexys3 v019	16 Kb	16 Kb	WT	4	No	No
Nexys3 v024	16 Kb	16 Kb	WT	4	No	No
Nexys3 v025	16 Kb	16 Kb	WT	4	No	No
Nexys3 v026	16 Kb	16 Kb	WT	4	No	No
Nexys3 v027	16 Kb	16 Kb	WT	4	No	No

Tabla 47: Resumen de configuraciones de caché en Digilent Nexys3

Plataforma	Caché de datos	Caché de instrucciones	Política de reemplazo	Tamaño de palabra	Caché Stream	Caché Victim
Atlys v001	16 Kb	16 Kb	WT	4	No	No
Atlys v002	32 Kb	32 Kb	WT	4	No	No
Atlys v003	64 Kb	64 Kb	WT	4	No	No
Atlys v007	32 Kb	32 Kb	WB	4	No	No
Atlys v008	32 Kb	32 Kb	WT	4	No	No
Atlys v009	32 Kb	32 Kb	WT	4	No	No
Atlys v010	32 Kb	32 Kb	WT	4	No	No
Atlys v017	64 Kb	64 Kb	WB	8	1	16 (8 I, 8 D)

Tabla 48: Resumen de configuraciones de caché en Digilent Atlys

